

Formalisation and implementation of Access control models

Mathieu Jaume
SPI LIP6, University Paris 6
Paris, France
Mathieu.Jaume@lip6.fr

Charles Morisset
SPI LIP6, University Paris 6
Paris, France
Charles.Morisset@lip6.fr

Abstract

Access control software must be based on a security policy model. Flaws in them may come from a lack of precision or some incoherences in the policy model or from inconsistencies between the model and the code. In this paper, we first present a formalisation of access control models based on the work on an algebra of security models by J.McLean [4]. Then, we describe the implementation of this framework and show how it can be used to obtain a particular security model: the Bell and La Padula security model. Last, as an example, we show how such a program can be integrated for secure databases. All our development is done within the Focal [11] programming environment which provides a language with object-oriented features allowing to write formal specifications, proofs and programs at the same level.

1. Introduction – Motivations

Security of information systems becomes a major problem for society and a well-established field of computer science. Research themes involve access control mechanisms, modeling of information flow and its applications to confidentiality policies [5, 6], mobile code security, cryptographic protocols, etc. The methods used to consider these questions are evolving, as are the ones used in safety areas, where ad-hoc and empirical approaches were progressively replaced by more formal methods. High levels of safety require that the requirement/specification phase is done using mathematical models, allowing mechanized proofs of the required properties. In the same way, assurance in system security asks for the use of true formal methods along the process of software development, starting at the specification level.

Computer information security is usually seen as a combination of three classes of properties: confidentiality (denying unauthorised accesses), integrity (denying unauthorised modifications of information) and availability

(denying unauthorised uses of ressources). To promote the conception of trusted systems, security evaluation criterias have been elaborated by government agencies, for example the Trusted Computer Security Evaluation Criterias (1983) (TCSEC), the Information Technology Security Evaluation Criterias (1991) (ITSEC) and the Common Criterias (1999), which are a collection of normative documents. These criterias provide both a framework for the software industry to ensure that software has been carefully designed and a referentie for its customers. A product evaluation and certification against the common criteria's framework is built according to two hypothesis. The first one is the "protection profile", that is, under which conditions the evaluated product is supposed to be used. The second one is its level of assurance, which is simply the level of trust the system can be granted, according to the development process. A good security level can be reached if the product is evaluated at an assurance level greater than EAL-5 against a convincing protection profile such as the one given in [8]. Such a profile requires the use of a mandatory formal policy model in order to achieve an acceptable level of confidentiality. Mandatory access control (MAC), contrary to discretionary access controls (DAC) such as access control list (ACL), is managed and enforced by the underlying system rather than by an authorized user.

Such a policy model must be precise and unambiguous and thus must be described in a mathematical formalism. This is the case for the general framework defined by J.McLean in [4]. Such a framework, in which various MAC security models allowing changes in security levels can be described, introduces joint access (of a set of subjects over an object) and provides means to compare security models.

However, even if having a mathematical model drawn by hand is a very serious way to increase confidence, this is not enough. First, nothing is said about how these abstract notions can be implemented: in this paper we present a way of formalising and implementing such notions. Furthermore, attempts to check proofs done by hand with a theorem prover has thrown away a lot of them. Often, the errors are introduced by points considered as evident details

or by forgotten cases. Last, even if proofs done by hand are correct, nothing formally ensures that the implementation meets the “formal” specification done “on the paper”.

Hence, in this paper, we briefly describe how Focal can be used to implement a generic framework for specifying and implementing access control models and then we show, as an example, that the well-known Bell and La Padula model can be obtained by instantiating this framework. Last, but not least, we illustrate the effective use of this program to manage access into a relational database.

Of course, our development may seem “heavy”. However, many access control models share some common definitions, properties and proofs and, as it is widely recognised, it seems desirable to provide an abstract generic framework in order to ease and speed implementations by reusing. Indeed, having an abstract formalism would allow to obtain an implementation well-suited for a given context just by instantiating parameters.

A very brief description of Focal The Focal project[9, 11]¹ attempts to build a new programming environment providing a language based on firm theoretical results, with clear semantics and which permits an efficient implementation – via translation to OCaml. It has functional and object-oriented features and provides means for the programmers to write formal proofs of their code, and to have them verified by a proof checker (Zenon and Coq).

In Focal, species are the nodes of the hierarchy of structures that makes up the library. A species can be seen as a set of methods, which are identified by their names. Each method can be either declared (primitive constants, operations and properties) or defined (implementation of operations and proofs). A species can inherit the declarations and definitions of one or several already defined species and is free to define or even redefine an inherited method (but must not change its type). Besides inheritance, the other important feature of Focal resides in the possibility to parameterize a species by another one.

A collection is built upon a completely defined species. This means that every method must be defined. In other words, in a collection, every operation has an implementation, and every theorem is formally proved. In addition, a collection is “frozen”. Namely, it cannot be used as a parent of a species in the inheritance graph.

2. Implementation of the “algebra of security” models

The algebra of security models has been introduced by J.McLean in [4]. It contains three levels of specifications: frameworks, models and systems.

¹<http://focal.inria.fr>

2.1. Frameworks

A framework \mathcal{F} is a quadruple $(\mathcal{S}, \mathcal{O}, \mathcal{L}^S, \mathcal{A})$ where \mathcal{S} is a set of subjects (users, programs, ...), \mathcal{O} is a set of objects (data, files, ...), $\mathcal{L}^S = (\mathcal{L}, \preceq, \gamma, \lambda)$ is a complete lattice of security levels and \mathcal{A} is a set of access modes (in the following, \mathcal{A}^* will be used to denote the set of finite subsets of \mathcal{A}). Each framework introduces a set \mathcal{S}^+ containing every set of subjects that can potentially perform a joint access over an object. Furthermore, as we will see, a framework needs to specify a set \mathcal{S}^{+*} of sets of sets of subjects (see models). Hence, we first define the species \mathbb{F} of framework containing the declaration of these sets.

Then, we introduce a species \mathbb{F}^g which inherits from \mathbb{F} in which \mathcal{S}^+ is specified as the finite powerset of \mathcal{S} without the empty set and \mathcal{S}^{+*} is specified as the finite powerset of \mathcal{S}^+ .

We also define a species \mathbb{F}_c , for coherent frameworks, which inherits from \mathbb{F} and specifies \mathcal{S}^{+*} as the set :

$$\{X \subseteq \mathcal{S}^+ \mid \forall y, z \in \mathcal{S}^+ (y \in X \wedge y \subseteq z) \Rightarrow z \in X\}$$

A coherent framework ensures that if a set of subjects is granted to change the security level of a subject or an object, then it can do it together with other subjects. This will be explained later when defining models. Note that at this level, nothing is said about \mathcal{S}^+ . Last, we introduce the species \mathbb{F}^c (resp. \mathbb{F}^s) which inherits from \mathbb{F}_c such that \mathcal{S}^+ is specified as the finite powerset of \mathcal{S} without the empty set (resp. as the set of singleton sets of subjects). Note that \mathbb{F}^s is clearly coherent (and then the property over \mathcal{S}^{+*} can be proved in this species) and is the most used in the literature (where no joint access are usually taken into account). The figure 1 describes the hierarchy of species of frameworks.

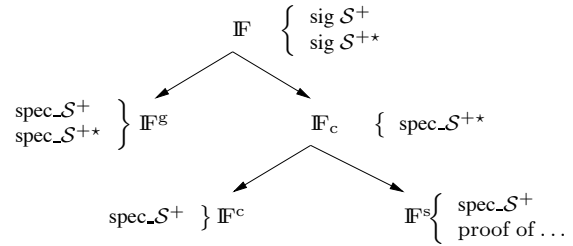


Figure 1. Hierarchy of frameworks

2.2. Model

Given a framework \mathcal{F} , a model is defined by two functions c_s and c_o , respectively from \mathcal{S} and \mathcal{O} , to \mathcal{S}^{+*} . These functions respectively associate with each subject and each object all the sets of subjects allowed to change the security level of the subject or the object. As the range of these functions is \mathcal{S}^{+*} , we see that by constraining \mathcal{S}^{+*} like we did

for coherent frameworks, we can express that if a (set of) subject is granted to change the security level of an entity, then it can also do it together with other subjects.

First, we define the species $\mathbb{M}_{\mathbb{F}}$ containing the declaration of c_s and c_o together with the declaration of two functions \sqcup and \sqcap with their specifications allowing to combine models. Let $M_1 = (\mathcal{F}, c_s^1, c_o^1)$ and $M_2 = (\mathcal{F}, c_s^2, c_o^2)$ be two models, then $M_1 \sqcup M_2 = (\mathcal{F}, c_s, c_o)$ with $\forall x \in \mathcal{S}, c_s(x) = c_s^1(x) \cup c_s^2(x)$ and $\forall x \in \mathcal{O}, c_o(x) = c_o^1(x) \cup c_o^2(x)$, and $M_1 \sqcap M_2 = (\mathcal{F}, c_s, c_o)$ with $\forall x \in \mathcal{S}, c_s(x) = c_s^1(x) \cap c_s^2(x)$ and $\forall x \in \mathcal{O} \quad c_o(x) = c_o^1(x) \cap c_o^2(x)$.

Since the functions c_s and c_o return a set of sets of subjects, inclusion over sets easily leads to an ordering between models defined as follows:

$$M_1 \leq M_2 \Leftrightarrow \left(\begin{array}{l} \forall x \in \mathcal{S} \quad c_s^1(x) \subseteq c_s^2(x) \\ \wedge \quad \forall x \in \mathcal{O} \quad c_o^1(x) \subseteq c_o^2(x) \end{array} \right)$$

Hence, we introduce the species $M_{\mathbb{F}}$ which inherits from $\mathbb{M}_{\mathbb{F}}$. $M_{\mathbb{F}}$ contains the declaration and specification of \leq , together with the proofs that \sqcap and \sqcup are respectively the operators of greatest lower bound (glb) and least upper bound (lub) for \leq . Since it can be easily shown that \sqcap and \sqcup are distributive, and that there exists two models which are the minimum and the maximum, the species $M_{\mathbb{F}}$ also inherits from the species of complete distributive lattice defined in the Focal library. The order \leq corresponds to the intuitive notion of stringness: for example, the minimum model is such that $c_s(s) = c_o(o) = \emptyset$ for all subject s and object o and then is clearly the more stringent model (no-one can change the security level of an entity).

Now, we introduce the species $M_{\mathbb{B}}$ which inherits from $M_{\mathbb{F}}$ and defines the complement of $\bar{\mathcal{M}} = (\mathcal{F}, \bar{c}_s, \bar{c}_o)$ of a model $\mathcal{M} = (\mathcal{F}, c_s, c_o)$ where:

$$\forall x \in \mathcal{S} \quad \bar{c}_s(x) = \mathcal{S}^+ \setminus c_s(x) \quad \wedge \quad \forall y \in \mathcal{O} \quad \bar{c}_o(y) = \mathcal{S}^+ \setminus c_o(y)$$

This species also inherits from the species of boolean algebra (defined in the Focal library). However, at this level, it's not possible to prove the required properties of boolean algebra since such a proof depends on the definitions of \mathcal{S}^+ and \mathcal{S}^{+*} which are not yet specified. So, we define two species $M_{\mathbb{F}^g}$ and $M_{\mathbb{F}^s}$ which inherits from $M_{\mathbb{B}}$: $M_{\mathbb{F}^g}$ is defined from a framework in \mathbb{F}^g while $M_{\mathbb{F}^s}$ is defined from a framework in \mathbb{F}^s . Furthermore, $M_{\mathbb{F}^s}$ also inherits from the species $M_{\mathbb{F}^c}$, since \mathbb{F}^s inherits from coherent frameworks. Last, we define the species $M_{\mathbb{F}^c}$ which inherits from $M_{\mathbb{F}^c}$ for models built upon a coherent framework. In this case, we just have a complete distributive lattice (the complement operation does not preserve the property for \mathcal{S}^{+*}). The figure 2 describes the hierarchy of species of models.

A more general ordering can be defined for models. Indeed, naming convention of subjects and objects may change from a model to another, and we want to be able

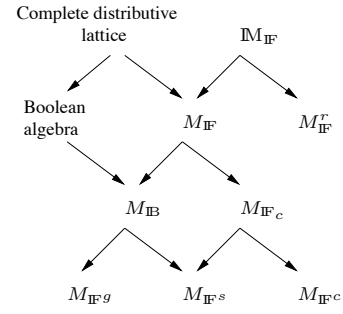


Figure 2. Hierarchy of models

to compare them independently of this naming convention. For this, J. McLean introduces the following relation: we say that $M_1 \triangleleft M_2$ iff there exists two one-to-one functions $I_s : \mathcal{S} \rightarrow \mathcal{S}$ and $I_o : \mathcal{O} \rightarrow \mathcal{O}$ such that:

$$\forall x \in \mathcal{S} \quad \forall y \in \mathcal{O} \quad \left(\begin{array}{l} c_s^1(x) \subseteq (I_s^+)^{-1}(c_s^2(I_s(x))) \\ \wedge \quad c_o^1(y) \subseteq (I_o^+)^{-1}(c_o^2(I_o(y))) \end{array} \right)$$

where $\forall X \in \mathcal{S}^{+*}, I_s^+(X) = \{K(Z) \mid Z \in X\}$ with $K(Z) = \{I_s(x) \mid x \in Z\}$. Actually, \triangleleft is just a preorder from which we can define an equivalence relation \equiv and an order relation \lesssim over equivalence classes of models. As \triangleleft generalises \leq , we would like to generalize the operators \sqcap and \sqcup . However, the set of equivalence classes of models is not a lattice but for each pair of models M_1 and M_2 , we can define the following set containing every least upper models of M_1 and M_2 :

$$\Psi^+(M_1, M_2) = \left\{ \begin{array}{l} M \in M_{\mathbb{F}}^r \mid \\ [M_1] \lesssim [M] \wedge [M_2] \lesssim [M] \wedge \\ \forall M' \left(\begin{array}{l} [M_1] \lesssim [M'] \\ \wedge [M_2] \lesssim [M'] \end{array} \right) \Rightarrow \neg([M'] < [M]) \end{array} \right\}$$

where $<$ is the strict order induced by \lesssim and where $[M]$ denotes the equivalence class of the model M . However, $\Psi^+(M_1, M_2)$ cannot be efficiently computed and we just have implemented the following operator:

$$M_1 \Delta M_2 = \left\{ \begin{array}{l} M \mid \exists M_3, M_4 \in M_{\mathbb{F}}^r \quad M_3 \equiv M_1 \\ \wedge \quad M_4 \equiv M_2 \wedge M = M_3 \sqcup M_4 \end{array} \right\}$$

which satisfies $\Psi^+(M_1, M_2) \subseteq \Delta(M_1, M_2)$. In a similar way, we also introduce the dual notions: Ψ^- and ∇ . All these notions are presented in details in [7] and are introduced in the species $M_{\mathbb{F}}^r$ which inherits from $\mathbb{M}_{\mathbb{F}}$.

2.3. System

Last comes the notion of system, where the security policy and the access control function must be defined. Given

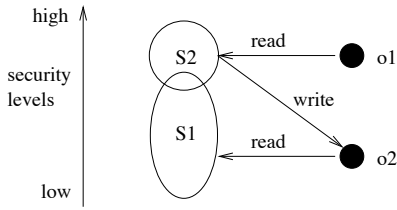


Figure 3. Violation of the \star -security

a framework and a model, a system can be viewed as an abstract state machine. The set Σ of states is defined as the product $\mathbb{F} \times \mathbb{A} \times \mathbb{A}$ where \mathbb{F} , is the product $\mathbb{F}_s \times \mathbb{F}_o$ with $\mathbb{F}_s = \{f_s : \mathcal{S} \rightarrow \mathcal{L}^S\}$ (security level associated to a subject) and $\mathbb{F}_o = \{f_o : \mathcal{O} \rightarrow \mathcal{L}^S\}$ (security level associated to an object), and $\mathbb{A} = \wp(\mathcal{S}^+ \times \mathcal{O} \times \mathcal{A}^*)$ is used to describe discretionary rights and current access of subjects over objects. Hence, a state $\sigma = ((f_s, f_o), d, m)$ defines mandatory rights (f_s and f_o , often called classification vector), discretionary rights (d) and current access (m). In order to specify joint access security policy, we extend f_s as follows:

$$\begin{aligned} f_s^\wedge : \mathcal{S}^+ \rightarrow \mathcal{L}^S & & f_s^\vee : \mathcal{S}^+ \rightarrow \mathcal{L}^S \\ f_s^\wedge(S) = \wedge \{f_s(s) \mid s \in S\} & & f_s^\vee(S) = \vee \{f_s(s) \mid s \in S\} \end{aligned}$$

At this level, we can define the security properties over states. Since we only have implemented the Bell and La Padula system, as an example of use of our development, in this species we only specify the three classical following properties (confinment and confidentiality):

- a state is said to be simple secure iff: $\forall(S, o, A) \in m, \text{read} \in A \Rightarrow f_s^\wedge(S) \succeq f_o(o)$
- a state is said to be \star -secure iff:

$$\left(\begin{array}{l} \forall S_1, S_2 \in \mathcal{S}^+ \forall o_1, o_2 \in \mathcal{O} \\ (S_1, o_1, \text{read}) \in m \\ \wedge (S_2, o_2, \text{write}) \in m \\ \wedge S_1 \cap S_2 \neq \emptyset \end{array} \right) \Rightarrow f_o(o_1) \preceq f_o(o_2)$$

This property prevents the copy of an object to a lower security level by a malicious subject (see figure 3). Note that this property is slightly different from [4], since for singleton frameworks, we want to obtain the classical definition of the “MAC- \star ” property of Bell and La Padula (this is not the case with the definition of McLean).

- a state is said to be ds-secure iff: $\forall S \in \mathcal{S}^+, \forall o \in \mathcal{O}, (S, o, x) \in m \Rightarrow (S, o, x) \in d$

A state satisfying these three properties is said to be secure.

Now we can introduce the species of system by defining a transition function $\tau : \mathcal{S}^+ \times \mathcal{R} \times \Sigma \rightarrow \mathcal{D} \times \Sigma$ between

states where \mathcal{R} is a set of requests and \mathcal{D} is a set of answers. In this species, we define a property over such transition functions: τ is said to be secure iff

$$\begin{aligned} \tau(S_1, r, \sigma_1) = (d, \sigma_2) \\ \Rightarrow \left(\begin{array}{l} \forall s \in \mathcal{S} \quad f_s^1(s) \neq f_s^2(s) \Rightarrow S_1 \in c_s(s) \\ \forall o \in \mathcal{O} \quad f_o^1(o) \neq f_o^2(o) \Rightarrow S_1 \in c_o(o) \end{array} \right) \end{aligned}$$

That is, if the security level of an entity has changed, then subjects that have performed this change were granted to do it (by the model).

Last, we introduce the species of secure systems which are systems with a secure initial state and a secure transition function mapping every secure state into a secure state (the well-known “Basic Security Theorem” of Bell and La Padula).

3. Applications

In this section, we briefly describe the instantiation of our framework in order to obtain the Bell and La Padula system, and then, we show how this program can be used to manage access in a relational database.

3.1. The Bell and La Padula system

In order to implement the Bell and La Padula system, we build a system from a framework in \mathbb{F}^s in which \mathcal{L}^S is the product lattice $T_c \times T_k$ where $T_c = (\mathcal{C}l, \leq, \sqcup_{cl}, \sqcap_{cl})$ is a lattice of classifications and $T_k = (\wp(\mathcal{K}), \subseteq, \cup, \cap)$ is the powerset lattice of the set \mathcal{K} of needs-to-know (note that the Focal library provides the implementation of product lattice) and $\mathcal{A} = \{\text{read, write, execute, append, control}\}$. The Bell and La Padula system is built from a model such that $c_s(s) = c_o(o) = \mathcal{S}^+$ for all subject s and object o^2 (in such a system, every transition function is secure). Now, in order to obtain a complete system, we just have to define requests, answers and the transition function. This has been done by following the paper of Bell and La Padula [3] and by reusing a formalisation within the Coq system [10] of a similar work [2]. Indeed, in [2], we have implemented a formal description of the Bell and La Padula model together with the formal proof of the “Basic Security Theorem”, checked by the theorem prover Coq.

3.2. Access control in a relational database

The system obtained in the above paragraph has been used to manage access in the Sqlite [1] relational database.

²Note that there exists a version of this system, called “Bell and La Padula system with tranquility” such that $c_s(s) = c_o(o) = \emptyset$ for all subject s and object o .

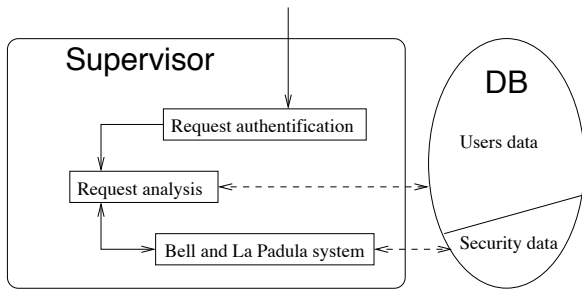


Figure 4. Database access control

Both security data (current access, discretionary rights, classifications and needs-to-know) and users data are stored in the database. The treatment of SQL queries is illustrated in figure 4. Given a SQL query submitted by an authenticated subject, our application analyses this query, generates a set of requests for the Bell and La Padula system, executes them, and then, depending on the answers obtained, performs or not the SQL query. For this, a “semantics” of SQL queries has been defined in terms of access. Note that queries are done by using the standard SQL syntax without any changes for users ; our program can be viewed as a “filter” of queries.

4. Conclusion – Future works

This development shows that even from a practical point of view, formal methods can be used to increase the security of software. Indeed, in this work, starting from a mathematical presentation of the algebra of security models, we have completely formalised this notion within the Focal environment and designed the architecture of its implementation.

The Focal language provides features allowing programs, specifications and proofs implementation in an incremental way. Indeed, inheritance and refinement mechanisms are powerful features which are particularly well-suited to develop a library for secure applications since they allow to make several refinements of a specification until providing an executable code. Furthermore, thanks to the (computer algebra) library available in Focal, all our species have been specified in a concise way. Thus, Focal avoids separately treating the formal modeling work and the development.

Moreover, Focal allows us to be very general during the development. For example, when defining the species of frameworks, models and systems, nothing is said about subjects and objects (no concrete datatype is associated to the abstract types of subjects and objects). This is only done at the end when we define a collection for a specific system (e.g. database access control). In this work, we have

obtained a complete specification of a system with access control and a transition function: we can apply it to any specific system, as a file manager or a database.

Another useful feature provided by Focal is the theorem prover Zenon. Indeed, such a prover allows to implement proofs in a very natural and declarative way without any background about the system Coq. Thus, it becomes possible for a large public to make proof within Focal.

Note that different kinds of users are involved in such a development. First, the Focal-developer supplies frameworks, models and systems, and possibly specific systems, like the Bell and La Padula system. This developer is in charge to give a sound specification and proofs that the implementation meets this specification. Then, the system administrator picks one framework, one model and one system, and gives an implementation of subjects and objects. The system administrator does not have any proof to do, but just has to use the correct formal system in regards to his specific system. Last, the system user can access to the data of the system with methods provided by the system administrator, which are generally the standard methods provided with this kind of specific system (e.g. `open_file`, `read_file`, etc), and of course these operations are sound in regards to the security policy.

We think that this development can be viewed as a basis for several extensions. For example, it could be interesting to implement others security models for access control (Chinese Wall, Roles Based Access Control, ...) in this framework. However, we also believe that some parts of this development can be reused in different contexts: for example, work on the implementation of management policy specification is currently under development.

Acknowledgements Many thanks to Julien Blond, Damien Doligez, Emmanuel Gureghian, Thérèse Hardin, Virgile Prevosto and Renaud Rioboo for enlightening discussions on this subject, as well as the anonymous referees for some very useful comments.

References

- [1] Sqlite: <http://www.sqlite.org/>.
- [2] E. Gureghian, T. Hardin, and M. Jaume. A full formalisation of the Bell and Lapadula security model. Technical Report 2003-007, Univ. Paris 6, LIP6, 2003.
- [3] L. LaPadula and D. Bell. Secure Computer Systems: A Mathematical Model. *Journal of Computer Security*, 4:239–263, 1996.
- [4] McLean. The algebra of security. In *Proc. IEEE Symposium on Security and Privacy*, pages 2–7. IEEE Computer Society Press, 1988.
- [5] J. McLean. Security models and information flow. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 180–187, 1990.
- [6] J. McLean. Security models. In J. Marciniak, editor, *Encyclopedia of Software Engineering*. J. Wiley & Sons, 1994.

- [7] C. Morisset. Formalisation et implantation d'un modèle de contrôle d'accès dans l'atelier Focal. Master's thesis, Université Paris 6, 2004.
- [8] NSA. Protection Profile For Multilevel Operating Systems In Environments Requiring Medium Robustness. Technical report, National Security Agency, 2001.
- [9] V. Prevosto and D. Doligez. Algorithms and proof inheritance in the foc language. *Journal of Automated Reasoning*, 29(3-4):337–363, dec 2002.
- [10] L. Project. *The Coq Proof Assistant Reference Manual Version 7*. INRIA-Rocquencourt, 2002.
- [11] R. Rioboo, D. Doligez, V. Prevosto, M. Jaume, M. Maarek, C. Dubois, S. Fechter, V. Ménissier-Morain, O. Pons, D. Delahaye, V. Viguié, and T. Hardin. *Focal, version 0.2 Tutorial and reference manual*. LIP6 – INRIA – CNAM, sept 2004. Distribution available at: <http://focal.inria.fr>.