

Certifying Airport Security Regulations using the Focal Environment

David Delahaye, Jean-Frédéric Étienne,
and Véronique Viguié Donzeau-Gouge

CEDRIC/CNAM, Paris, France,
David.Delahaye@cnam.fr, etien_je@auditeur.cnam.fr,
donzeau@cnam.fr

Abstract. We present the formalization of regulations intended to ensure airport security in the framework of civil aviation. In particular, we describe the formal models of two standards, one at the international level and the other at the European level. These models are expressed using the Focal environment, which is also briefly presented. Focal is an object-oriented specification and proof system, where we can write programs together with properties which can be proved semi-automatically. We show how Focal is appropriate for building a clean hierarchical specification for our case study using, in particular, the object-oriented features to refine the international level into the European level and parameterization to modularize the development.

1 Introduction

The security of civil aviation is governed by a series of international standards and recommended practices that detail the responsibilities of the various stakeholders (states, operators, agents, etc). These documents are intended to give the specifications of procedures and artifacts which implement security in airports, aircraft and air traffic control. A key element to enforce security is the conformance of these procedures and artifacts to the specifications. However, it is also essential to ensure the consistency and completeness of the specifications. Standards and recommended practices are natural language documents (generally written in English) and their size may range from a few dozen to several hundred pages. Natural language has the advantage of being easily understood by a large number of stakeholders, but practice has also shown that it can be interpreted in several inconsistent ways by various readers. Moreover, it is very difficult to process natural language documents automatically in the search for inconsistencies. When a document has several hundred pages, it is very difficult to ensure that the content of a particular paragraph is not contradicted by some others which may be several dozen pages from the first one.

This paper aims to present the formal models of two standards related to airport security in order to study their consistency: the first one is the international standard Annex 17 [7] (to the Doc 7300/8) produced by the International Civil Aviation Organization (ICAO), an agency of the United Nations; the second one

is the European standard Doc 2320 [2] (a public version of the Doc 30, which has a restricted access status) produced by the European Civil Aviation Conference (ECAC) and which is supposed to refine the first one at the European level. More precisely, from these models, we can expect:

1. to detect anomalies such as inconsistencies, incompleteness and redundancies or to provide evidence of their absence;
2. to clarify ambiguities and misunderstandings resulting from the use of informal definitions expressed in natural language;
3. to identify hidden assumptions, which may lead to shortcomings when additional explanations are required (e.g. in airport security programmes);
4. to make possible the rigorous assessment of satisfaction for a concrete regulation implementation and w.r.t. the requirements.

This formalization was completed in the framework of the EDEMOI¹ [8] project, which aims to integrate and apply several requirements engineering and formal methods techniques to analyze regulation standards in the domain of airport security. The methodology of this project may be considered as original in the sense that it tries to apply techniques, usually reserved to critical software, to the domain of regulations (in which no implementation is expected). The project used a two-step approach. In the first step, standards described in natural language were analyzed in order to extract security properties and to elaborate a conceptual model of the underlying system [5]. The second step, which this work is part of, consists in building a formal model and to analyze/verify the model by different kinds of formal tools. In this paper, we describe two formal models of the two standards considered above, which have been carried out using the Focal [12] environment, as well as some results that have been analyzed from these models.

Another motivation of this paper is to present the Focal [12] (previously Foc) environment, developed by the Focal team, and to show how this tool is appropriate to model this kind of application. The idea is to assess and validate the design features as well as the reasoning support mechanism offered by the Focal specification and proof system. In our case study, amongst others, we essentially use the features of inheritance and parameterization. Inheritance allows us to get a neat notion of refinement making incremental specifications possible; in particular, the refinement of the international level by the European level can be expressed naturally. Parameterization provides us with a form of polymorphism so that we can factorize parts of our development and obtain a very modular specification. Finally, regarding the reasoning support, the first-order automated theorem-prover of Focal, called Zenon, bring us an effective help by automatically discharging most of the proofs required by the specification.

The paper is organized as follows: first, we give a brief description of the Focal language with its main structures and features; next, we present our case study, i.e. the several standards regulating security in airports and in particular,

¹ The EDEMOI project is supported by the French National "Action Concertée Incitative Sécurité Informatique".

those we chose to model; finally, we describe the global formalization made in Focal, as well as the properties that could be analyzed and verified.

2 The Focal environment

2.1 What is Focal?

Focal [12], initiated by T. Hardin with R. Rioboo and S. Boulmé, is a language in which it is possible to build applications step by step, going from abstract specifications, called *species*, to concrete implementations, called *collections*. These different structures are combined using inheritance and parameterization, inspired by object-oriented programming; moreover, each of these structures is equipped with a carrier set, providing a typical algebraic specification flavor. Moreover, in this language, there is a neat separation between the activities of programming and proving. A compiler was developed by V. Prevosto for this language, able to produce Ocaml [11] code for execution, Coq [10] code² for certification, but also code for documentation (generated by means of structured comments). More recently, D. Doligez provided a first-order automated theorem prover, called Zenon, which helps the user to complete his/her proofs in Focal through a declarative-like proof language. This automated theorem prover can produce pure Coq proofs, which are reinserted in the Coq specifications generated by the Focal compiler and fully verified by Coq.

2.2 Specification: species

The first major notion of the Focal language is the structure of *species*, which corresponds to the highest level of abstraction in a specification. A species can be roughly seen as a list of attributes and there are three kinds of attributes:

- the carrier type, called *representation*, which is the type of the entities that are manipulated by the functions of the species; representations can be either abstract or concrete;
- the functions, which denote the operations allowed on the entities; the functions can be either *definitions* (when a body is provided) or *declarations* (when only a type is given);
- the properties, which must be verified by any further implementation of the species; the properties can be either simply properties (when only the proposition is given) or theorems (when a proof is also provided).

More concretely, the general syntax of a species is the following:

² Here, Coq is only used as a proof checker, and not to extract, from provided proofs and using its Curry-Howard isomorphism capability, Ocaml programs, which are directly generated from Focal specifications.

```

species <name> =

    rep [= <type>];           (* abstract/concrete
                               representation *)

    sig <name> in <type>;     (* declaration *)
    let <name> = <body>;      (* definition *)

    property <name> : <prop>; (* property *)
    theorem <name> : <prop>  (* theorem *)
    proof : <proof>;

end

```

where <name> is simply a given name, <type> a type expression (mainly typing of core-ML without polymorphism but with inductive types), <body> a function body (mainly core-ML with conditional, pattern-matching and recursion), <prop> a (first-order) proposition and <proof> a proof (expressed in a declarative style and given to Zenon). In the type language, the specific expression **self** refers to the type of the representation and may be used everywhere except when defining a concrete representation.

As said previously, species can be combined using (multiple) inheritance, which works as expected. It is possible to define functions that were previously only declared or to prove properties which had no provided proof. It is also possible to redefine functions previously defined or to reprove properties already proved. However, the representation cannot be redefined and functions as well as properties must keep their respective types and propositions all along the inheritance path. Another way of combining species is to use parameterization. Species can be parameterized either by other species or by entities from species. If the parameter is a species, the parameterized species only has access to the interface of this species, i.e. only its abstract representation, its declarations and its properties. These two features complete the previous syntax definition as follows:

```

species <name> (<name> is <name>, <name> in <name>, ...)
    inherits <name>, <name> (<pars>), ... = ...

end

```

where <pars> is a list of <name> and denotes the names which are used as parameters. When the parameter is a species, the keyword is **is**, when it is an entity of a species, the keyword is **in**.

To better understand this notion of species, let us give a small example:

Example 1 (Finite stacks). To formalize finite stacks, an abstract way is to specify stacks (possibly infinite) first, and to refine them as finite stacks afterwards. The specification of stacks might be the following:

```

species stack (typ is setoid) inherits setoid =

  sig empty in self;
  sig push in typ → self → self;
  sig pop in self → self;
  sig last in self → typ;
  let is_empty (s) = !equal (s, !empty);

  property ie_empty : !is_empty (!empty);
  property ie_push : all e in typ, all s in self,
    not (!is_empty (!push (e, s))); ...

end

```

where setoid is a predefined species representing a non-empty set with an equality (in the first line, the parameter and the inheritance from setoid show respectively that we want to be able to compare two items of a stack, but also two stacks), the "!" notation is equivalent to the common dot notation of message sending in object-oriented programming (**self** is the default species when there is no receiver species indicated; e.g. !empty is for **self**!empty).

Next, before specifying finite stacks, we can be more modular and formalize the notion of finiteness separately as follows:

```

species is_finite (max in int) inherits basic_object =

  sig size in self → int;
  property size_max : all s in self, #int_leq (!size (s), max);

end

```

where basic_object is a predefined species supposed to be the root of every Focal hierarchy, int the predefined type of integers and "#int_" the prefix of operations over the type int. Here, we can remark that the species is parameterized by an entity of a species and not by a species.

Finally, we can formalize finite stacks using a multiple inheritance from the species stack and is_finite:

```

species finite_stack (typ is setoid, max in int)
  inherits stack (typ), is_finite (max) =

  let is_full (s) = #int_eq (!size (s), max);

  property size_empty : #int_eq (!size (!empty), 0);
  property size_push : all e in typ, all s in self, not (!is_full (s)) →
    #int_eq (!size (!push (e, s)), #int_plus (!size (s), 1)); ...

end

```

2.3 Implementation: collection

The other main notion of the Focal language is the structure of *collection*, which corresponds to the implementation of a specification. A collection implements a species in such a way that every attribute becomes concrete: the representation must be concrete, functions must be defined and properties must be proved. If the implemented species is parameterized, the collection must also provide implementations for these parameters: either a collection if the parameter is a species or a given entity if the parameter denotes an entity of a species. Moreover, a collection is seen (by the other species and collections) through its corresponding interface; in particular, the representation is an abstract data type and only the definitions of the collection are able to manipulate the entities of this type. Finally, a collection is a terminal item and cannot be extended or refined by inheritance. The syntax of a collection is the following:

```
collection <name> implements <name> (<pars>) = ... end
```

We will not detail examples of collections here since our formalization (see Section 4) does not make use of them. Actually, the airport security regulations considered in this paper are rather abstract and do not expect any implementation. Regarding our previous example of finite stacks, a corresponding collection will have to provide a concrete representation (using lists for example), definitions for only declared functions (`empty`, `push`, `pop`, `last`) and proofs for properties (`ie_empty`, `ie_push`, etc). For complete examples of collections, the reader can refer to the standard library of Focal (see Section 2.5).

2.4 Certification: proving with Zenon

The certification of a Focal specification is ensured by the possibility of proving properties. To do so, a first-order automated theorem prover, called *Zenon* and based on the tableau method, helps us to complete the proofs. Basically, there are two ways of providing proofs to *Zenon*: the first one is to give all the properties (proved or not) and definitions needed by *Zenon* to build a proof with its procedure; the second one is to give additional auxiliary lemmas to help *Zenon* to find a proof. In the first option, *Zenon* must be strong enough to find a proof with only the provided properties and definitions; the second option must be considered when *Zenon* needs to be helped a little more or when the user likes to present his/her proof in a more readable form. In the first option, proofs are described as follows:

```
theorem <name> : <prop>  
proof : by <props> def <defs>;
```

where `<props>` is a list of properties and `<defs>` a list of definitions.

The proof language of the second option is inspired by a proposition by L. Lamport [6], which is based on a practical and hierarchical structuring of proofs using number labels for proof depth. We do not describe this language

here but some examples of use can be found in the formalization of our case study (see Section 4.4 to get the development).

Let us describe a small proof in our example of finite stacks:

Example 2 (Finite stacks). In the species stack, we can notice that with the definition of `is_empty`, Property `ie_empty` can already be proved in the following way:

```
theorem ie_empty : !is_empty (!empty)
proof : by !equal_reflexive def !is_empty;
```

where `equal_reflexive` is the property of reflexivity for equality, which is inherited from the species setoid.

This proof uses the definition of `is_empty`, which means that any redefinition of `is_empty` in any further inheritance invalidates this proof (which has to be completed again using the new definition). Thus, w.r.t. usual object-oriented programming, redefinitions may have some additional effects since they directly influence the proofs in which they are involved.

2.5 Further information

For additional information regarding Focal, the reader can refer to [3], as well as to the Focal Web site: <http://focal.inria.fr/>, which contains the Focal distribution (compiler, Zenon and other tools), the reference manual, a tutorial, some FAQs and also some other references regarding, in particular, Focal's formal semantics (e.g. see S. Boulmé and S. Fechter's PhD theses).

3 Case study: airport security regulations

The primary goal of the international standards and recommended practices regulating airport security is to safeguard civil aviation against acts of unlawful interference. These normative documents detail the roles and responsibilities of the various stake-holders and pinpoint a set of security measures (as well as the ways and means to implement them) that each airport serving civil aviation has to comply with. In addition, the entire regulatory system is organized in a hierarchical way, where each level has its own set of regulatory documents that are drafted and maintained by different bodies. At the international level, Annex 17 [7] of the International Civil Aviation Organization (ICAO) lays down the general principles and recommended practices to be adopted by each member state. It is refined at the European level by the Doc 2320 [2] of the European Civil Aviation Conference (ECAC), where the standard is made more detailed and more precise. At the national level, each member state has to establish and implement a national civil aviation security programme in compliance with international standards and national laws. Finally, at the airport level, the national and international standards are implemented by an airport security programme.

All these documents are written in natural language and due to their voluminous size, it is difficult to manually assess the consistency of the entire regulatory system. Moreover, informal definitions tend to be inaccurate and may be interpreted in various inconsistent ways by different readers. Consequently, it may happen that two inspectors visiting the same airport at the same time reach contradictory conclusions about its conformity. However, these documents have the merit of being rigorously structured. Ensuring their consistency and completeness while eliminating any ambiguity or misunderstanding is a significant step towards the reinforcement of airport security.

3.1 Scope delimitation

After a deep study of the above-mentioned documents and several consultations with the ICAO and ECAC, we decided to take as a starting point the preventive security measures described in Chapter 4 of Annex 17. Chapter 4 begins by stating the primary goal to be fulfilled by each member state, which is:

4.1 Each Contracting State shall establish measures to prevent weapons, explosives or any other dangerous devices, articles or substances, which may be used to commit an act of unlawful interference, the carriage or bearing of which is not authorized, from being introduced, by any means whatsoever, on board an aircraft engaged in international civil aviation.

Basically, this means that acts of unlawful interference can be avoided by preventing unauthorized dangerous objects from being introduced on board aircraft³. To be able to achieve this goal, the member states have to implement a set of preventive security measures, which are classified in Chapter 4 according to six specific situations that may potentially lead to the introduction of dangerous objects on board. These are namely:

- persons accessing restricted security areas and airside areas (A17, 4.2);
- taxiing and parked aircraft (A17, 4.3);
- ordinary passengers and their cabin baggage (A17, 4.4);
- hold baggage checked-in or taken in custody of airline operators (A17, 4.5);
- cargo, mail, etc, intended for carriage on commercial flights (A17, 4.6);
- special categories of passengers like armed personnel or potentially disruptive passengers that have to travel on commercial flights (A17, 4.7).

At the lower levels of the regulatory hierarchy, the security measures are refined and detailed in such a way as to preserve the decomposition presented above. This structure allowed us to easily identify the relation between the different levels of refinement. Due to the restricted access nature of some of the regulatory documents, the formalization presented in Section 4 only considers Chapter 4 of Annex 17 and some of the refinements proposed by the European Doc 2320. Moreover, for simplification reasons, we do not cover the security measures 4.3 and 4.6.

³ Note that the interpretation given to the quoted paragraph may appear wrong to some readers. In fact, Paragraph 4.1 is ambiguous as it can be interpreted in two different ways (see Section 4.4 for more details).

3.2 Modeling challenges

Modeling the regulations governing airport security is a real world problem and is therefore a good exercise to identify the limits of the inherent features of the Focal environment. Moreover, the ultimate objective of such an application is not to produce certified code but rather to provide an automated support for the analysis of the regulatory documents. For this case study, the formalization needs to address the following modeling challenges:

1. the model has to impose a structure that facilitates the traceability and maintainability of the normative documents. Moreover, through this structure, it should be possible to easily identify the impact of a particular security measure on the entire regulatory system;
2. the model must make the distinction between the security measures and the ways and means of implementing them. Most of the security measures are fairly general and correspond to reachable objectives. However, their implementation may differ from one airport to another due to national laws and local specificities;
3. for each level of the regulatory hierarchy, the model must determine (through the use of automated reasoning support tools) whether or not the fundamental security properties can be derived from the set of prescribed security measures. This will help to identify hidden assumptions made during the drafting process. In addition, the model has to provide evidence that the security measures defined at refined levels are not less restrictive than those at higher levels.

4 Formalization

4.1 Model domain

In order to formalize the meaning of the preventive security measures properly, we first need to identify the subjects they regulate, together with their respective properties/attributes and the relationships between them. It is also essential to determine the hierarchical organization of the identified subjects in order to effectively factorize functions and properties during the formalization process. This is done by determining the dependencies between the security measures, w.r.t. the definitions of terms used in the corresponding normative document. For example, let us consider the following security measure described in Chapter 4 of Annex 17:

4.4.1 Each Contracting State shall establish measures to ensure that originating passengers of commercial air transport operations and their cabin baggage are screened prior to boarding an aircraft departing from a security restricted area.

To be able to formalize this security measure, it is obvious that we will have to define the subjects *originating passenger*, *cabin baggage*, *aircraft* and *security restricted area*, together with the relations between them. Moreover, we will need to define appropriate attributes for the *originating passenger* subject to characterize the state of being *screened* and of being *on board*. Finally, to complete the formalization, we will have to specify the integrity constraints induced by the regulation (e.g. screened passengers are either in security restricted areas or on board aircraft). The hierarchies of subjects obtained after analyzing all the preventive security measures of Annex 17 are represented by a Focal model, where each subject is a species. For instance, the Focal model for airside persons is given in Figure 1 (where nodes are species and arrows inheritance relations s.t. $A \leftarrow B$ means species B inherits from A).

For possible extensions during the refinement process, the *representation* of the species is left undefined (abstract) and their functions are only declared. Moreover, since we are not concerned with code generation, our formalization does not make use of collections. For example, the following species corresponds to the specification of the *cabin person* subject:

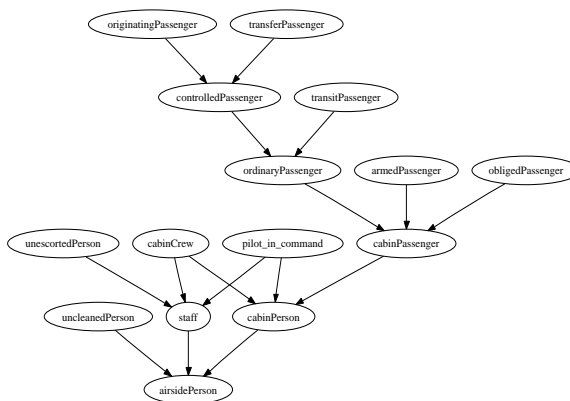


Fig. 1. Hierarchy for airside persons in Annex 17.

```

species cabinPerson (obj is object, obj_set is basic_set (obj),
    do is dangerousObject, do_set is basic_set (do),
    wp is weapon, wp_set is basic_set (wp), id is identity,
    c_luggage is cabinLuggage (obj, obj_set, do, do_set, wp, wp_set),
    cl_set is basic_lset (obj, obj_set, do, do_set, wp, wp_set, c_luggage))
inherits airsidePerson (obj, obj_set, do, do_set, wp, wp_set, id) =

```

```

sig embarked in self -> bool;
sig get_cabinLuggage in self -> cl_set;

```

```

property invariant_weapons : all w in wp, all s in self,
    wp_set!member (w, !get_weapons (s)) -> not (wp!inaccessible (w));

```

end

The species *cabinPerson* specifies the common functions and properties for the different types of persons who are eligible to travel on board an aircraft. In order to specify the relations between cabin persons and the different items they can have access to during flight time, the species *cabinPerson* is parameterized with the species *object*, *dangerousObject*, *weapon* and *cabinLuggage*. The parameters *obj_set*, *do_set*, *wp_set* and *cl_set* describe the sets of the previously

identified items; they are introduced to express the fact that a cabin person can own more than one item at a time. Since most of these relations are already specified in the species `airsidePerson`, they are inherited automatically. The function `get_cabinLuggage` is only introduced to make accessible the set of cabin luggage associated to a given instance of `cabinPerson`. Property `invariant_weapons` is a typical example of integrity constraints imposed by the regulation. It states that when weapons are carried by cabin persons, they are *by default* considered to be accessible during flight time.

4.2 Annex 17: preventive security measures

As said in Section 3.2, the formal model needs to impose a certain structure that will facilitate the traceability and maintainability of the normative documents. To achieve this purpose, our model follows the structural decomposition proposed in Chapter 4 of Annex 17 (using inheritance), while taking into account the dependencies between the preventive security measures. In our model, since most of the security measures correspond to reachable objectives, they are defined as invariants and each airport security programme must provide procedures which satisfy these invariants. However, when the security measures describe actions to be taken when safety properties are violated, a procedural approach is adopted. The consistency and completeness of the regulation are achieved by establishing that the fundamental security property, defined in Paragraph 4.1 of Annex 17, is satisfied by all the security measures, while ensuring their homogeneity. The general structure of the Annex 17 model is represented in Figure 2.

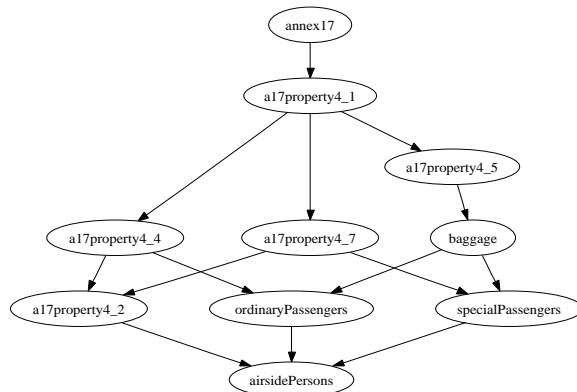


Fig. 2. Structure of Annex 17.

dependencies are defined according to the hierarchical organization of the subjects they regulate. The fundamental security property is defined in species `a17property4_1`. It is at this level that the set of on board objects is defined. Finally, the theorems establishing the consistency and completeness of the regulation are defined in the species `annex17`.

The species `airsidePersons`, `ordinaryPassengers`, `specialPassengers` and `baggage` introduce the set domain of the subjects presented in Section 4.1 as well as their relational constraints (e.g. two passengers cannot have the same luggage). The preventive security measures are formalized in species `a17property4_2`, `a17property4_4`, `a17property4_5`, `a17property4_7` and their dependencies

Security measures related to ordinary passengers As an example, we can focus on Property 4.4 of Annex 17 related to security measures for ordinary passengers. This property is divided into four sub-properties and, for example, we can describe how Property 4.4.1 (cited in Section 4.1) was formalized:

Example 3 (Property 4.4.1). Security measure 4.4.1 states that originating passengers and their cabin baggage should be screened prior to boarding an aircraft. In species `a17property4_4`, this statement is formalized as follows:

```
property property_4_4_1 : all p in op, all s in self,
  op_set!member (p, !originatingPassengers (s)) ->
  op!embarked (p) -> op!screened (p);
```

where `p` represents an originating passenger and `s` the current state of species `a17property4_4`. It should be noted that the scope of the boolean function `screened` extends to cabin baggage as well, since cabin baggage remains with its owners throughout the boarding process. The fact of being a screened ordinary passenger is defined in the species `controlledPassengers` (see Figure 1) as follows:

```
property invariant_screened : all s in self,
  !screened (s) -> wp_set!is_empty (!get_weapons (s)) and
  wp_set!is_empty (cl_set!get_weapons (!get_cabinLuggage (s))) and
  all o in do, do_set!member (o, !get_dangerousObjects (s)) or
  do_set!member (o, cl_set!get_dangerousObjects
    (!get_cabinLuggage (s))) -> dolis_authorized (o);
```

where `s` represents a `controlledPassenger`. Property `invariant_screened` states that if a passenger is screened, he/she does not have any weapons and if the passenger does have a dangerous object (other than weapons), it is authorized. A similar property also exists for Property 4.4.2 (which concerns transfer passengers) and could be factorized via the parameterization mechanism of `Focal`.

From this property and the three others (4.4.2, 4.4.3 and 4.4.4), we can prove the global property 4.4 that ordinary passengers admitted on board an aircraft do not have any unauthorized dangerous objects. This intermediate lemma is used afterwards when proving the consistency of the fundamental security property (4.1) w.r.t. the preventive security measures.

Consistency of Annex 17 Once we completed the formalization for each of the different categories of preventive security measures and derived the appropriate intermediate lemmas, we can consider Paragraph 4.1 (see Section 3.1) of Annex 17. It is formalized as follows in species `a17property4_1`:

```
property property_4_1 : all a in ac, all s in self,
  ac_set!member (a, !departureAircraft (s)) ->
  (all o in do, do_set!member (o, !onboardDangerousObjects (a, s)) ->
    do!is_authorized (o)) and
  (all o in wp, wp_set!member (o, !onboardWeapons (a, s)) ->
    wp!is_authorized (o));
```

where a represents an aircraft. This states that dangerous objects are admitted on board a departing aircraft only if they are authorized. In addition, the set of on board objects for each departing aircraft is defined according to the different types of cabin persons (together with their cabin luggage) and according to the different types of hold baggage loaded into the aircraft. This correlation is necessary since it will allow us to establish the following consistency theorem:

theorem consistency : !property_4_2 -> !property_4_4 ->
!property_4_5 -> !property_4_7 -> !property_4_1
proof : by do_set!union1, wp_set!union1 def !property_4_2, !property_4_4,
!property_4_5, !property_4_7, !property_4_1;

where property_4_2, property_4_4, property_4_5 and property_4_7 correspond to the intermediate lemmas defined for each category of preventive security measures. The purpose of Theorem consistency is to verify whether the fundamental security property can be derived from the set of preventive security measures. This allowed us to identify some hidden assumptions done during the drafting process (see Section 4.4). However, this theorem does not guarantee the absence of contradictions in the regulation. A way to tackle this problem is to try to derive False from the set of security properties and to let Zenon work on it for a while. If the proof succeeds then we have a contradiction, otherwise we can only have a certain level of confidence.

4.3 Doc 2320: some refinements

The document structure of Doc 2320 follows the decomposition presented in Chapter 4 of Annex 17. Refinement in Doc 2320 appears at two levels. At the subject level, the refinement consists in enriching the characteristics of the existing subjects or in adding new subjects. At the security property level, the security measures become more precise and sometimes more restrictive. The verification of the consistency and completeness of Doc 2320 is performed in the same way as for Annex 17 (see the modeling described in Section 4.2).

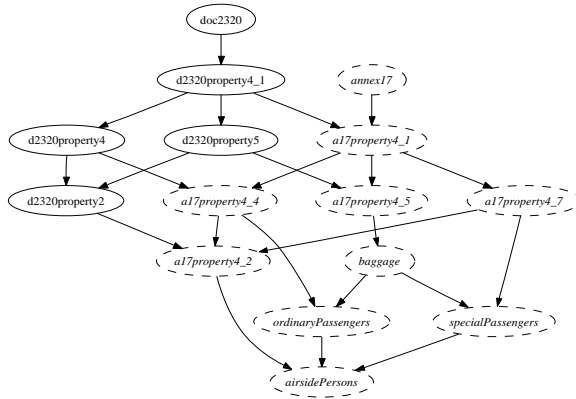


Fig. 3. Structure of Doc 2320.

However, since Doc 2320 refines Annex 17, an additional verification is required to show that the security measures that it describes do not invalidate the ones defined in Annex 17. Thus, in addition to consistency proofs, another kind of proofs appears, that are refinement proofs. The model structure obtained for Doc 2320 is described in Figure 3 (where the existing species coming from

Annex 17 are distinguished with dashed nodes). As can be seen, the refinement is performed in such a way as to preserve the dependencies between the security measures. Moreover, it can be observed that unlike species a17property4_2, a17property4_4 and a17property4_5, species a17property4_7 does not have a Doc 2320 counterpart. This is because, in Doc 2320, no mention to special categories of passengers is made. We assume that in this case, the international standard still prevails.

A refinement example In Doc 2320, Property 4.4.1 of Annex 17 is refined by Property 4.1.1, which states that originating passengers are either searched by hand or screened prior to boarding an aircraft. In species d2320property4, this statement is formulated as follows:

```
property d2320property_4_1_1 : all p in op, all s in self,
  op_set!member (p, !originatingPassengers (s)) ->
  op!embarked (p) -> op!screened (p) or op!handSearched (p);
```

To prove that Property d2320property_4_1_1 does not invalidate Property property_4_4_1, the following theorem is used:

```
theorem refinement : !d2320property_4_1_1 -> !property_4_4_1
proof : by oplinvariant_handSearched, oplinvariant_screened
def !d2320property_4_1_1, !property_4_4_1;
```

The above theorem is provable since in species controlledPassenger2320, which is a refined version of species controlledPassenger, the boolean function handSearched is characterized by the same properties than the boolean function screened (e.g. Property invariant_screened).

4.4 Analyses and results

An example of ambiguity As seen in Section 3, Paragraph 4.1 of Annex 17 is very important as it states the primary goal of the preventive security measures to be implemented by each member state. However, it appears to be ambiguous since it can be interpreted in two different ways: either dangerous objects are *never* authorized on board or they are admitted on board *only if* they are authorized. According to the ICAO, the second interpretation is the correct one as Paragraph 4.1 needs to be considered in the general context of the regulation to clarify this ambiguity.

Hidden assumptions In trying to demonstrate that Paragraph 4.1 of Annex 17 is consistent w.r.t. the set of preventive security measures, we discovered, for instance, the following hidden assumptions:

1. since disruptive passengers who are obliged to travel are generally escorted by law enforcement officers, they are considered not to have any dangerous objects in their possession;

2. unlike other passengers, transit passengers are not subjected to any specific security control but should be protected from unauthorized interference at transit spots. This implies that they are considered to be secure and hence do not have any unauthorized dangerous objects.

Development The entire formalization takes about 10000 lines of Focal code, with in particular, 150 species and 200 proofs. It took about 2 years to be completed. The development is freely available (sending a mail to the authors) and can be compiled with the latest version of Focal (0.3.1).

5 Conclusion

Summary A way to improve security is to produce high quality standards. The formal models of Annex 17 and Doc 2320 regulations, partially described in this paper, tend to bring an effective solution in the specific framework of airport security. From these formalizations, some properties could be analyzed and in particular, the notion of consistency. This paper also aims to emphasize the use of the Focal language, which provides a strongly typed and object-oriented formal development environment. The notions of inheritance and parameterization allowed us to build the specifications in an incremental and modular way. Moreover, the Zenon automated theorem prover (provided in Focal) discharged most of the proof obligations automatically and appeared to be very appropriate when dealing with abstract specifications (i.e. with no concrete representation).

Related work Currently, models of the same regulations, by D. Bert and his team, are under development using B [1] in the framework of the EDEMOI project. In the near future, it could be interesting to compare the two formal models (in Focal and B) rigorously in order to understand if and how the specification language influences the model itself. It should be noted that the same results (see Section 4.4) were obtained from this alternative formalization, since some of these results were already analyzed before the formalization itself (during the conception step). Very close to the EDEMOI project is the SAFEE project [9], funded by the 6th Framework Programme of the European Union (FP6) and which aims to use similar techniques for security but on board the aircraft. Regarding similar specifications in Focal, we must keep in mind that the compiler is rather recent (4/5 years at most) and efforts have been essentially provided, by R. Rioboo, to build a Computer Algebra library, which is currently the standard library of Focal. However, some more applicative formalizations are under development like certified implementations of security policies [4] by M. Jaume and C. Morisset.

Future work We plan to integrate a test suite into this formalization using an automatic generation procedure (working from a Focal specification) and using stubs for abstract functions (i.e. only declared). Amongst other things, this will

allow us to imagine and build attack scenarios which, at least in this context, appear to be quite interesting for official certification authorities. Such an automatic procedure is currently work in progress, by C. Dubois and M. Carlier, but is still limited (to universally quantified propositions) and needs to be extended to be applied to our development. We also plan to produce UML documents automatically generated from the Focal specifications and which is an effective solution to interact with competent organizations (ICAO, ECAC). Such a tool has been developed by J. F. Étienne but has to be completed to deal with all the features of Focal. Regarding the Focal language itself, some future evolutions might be appropriate, in particular, the notion of subtyping (there is a notion of subspecies but it does not correspond to a relation of subtyping), but which still needs to be specified in the case of properties. Also, it might be necessary to integrate temporal features in order to model behavioral properties, since in fact, our formalization, described in this paper, just shows a static view of the specified regulations.

References

1. J. R. Abrial. *The B Book, Assigning Programs to Meanings*. Cambridge University Press, Cambridge (UK), 1996. ISBN 0521496195.
2. The European Civil Aviation Conference. *Regulation (EC) N° 2320/2002 of the European Parliament and of the Council of 16 December 2002 establishing Common Rules in the Field of Civil Aviation Security*, December 2002.
3. C. Dubois, T. Hardin, and V. Vigié Donzeau-Gouge. Building Certified Components within Focal. In *Symposium on Trends in Functional Programming (TFP)*, volume 5, pages 33–48, Munich (Germany), November 2004. Intellect (Bristol, UK).
4. M. Jaume and C. Morisset. Formalisation and Implementation of Access Control Models. In *Information Assurance and Security (IAS), International Conference on Information Technology (ITCC)*, pages 703–708, Las Vegas (USA), April 2005. IEEE CS Press.
5. R. Laleau, S. Vignes, Y. Ledru, M. Lemoine, D. Bert, V. Vigié Donzeau-Gouge, and F. Peureux. Application of Requirements Engineering Techniques to the Analysis of Civil Aviation Security Standards. In *International Workshop on Situational Requirements Engineering Processes (SREP), in conjunction with the 13th IEEE International Requirements Engineering Conference*, Paris (France), August 2005.
6. L. Lamport. How to Write a Proof. *American Mathematical Monthly*, 102(7):600–608, August 1995.
7. The International Civil Aviation Organization. *Annex 17 to the Convention on International Civil Aviation, Security - Safeguarding International Civil Aviation against Acts of Unlawful Interference, Amendment 11*, November 2005.
8. The EDEMOI project, 2003. <http://www-lsr.imag.fr/EDEMOI/>.
9. The SAFEE project, 2004. <http://www.safee.reading.ac.uk/>.
10. The Coq Development Team. Coq, *version 8.0*. INRIA, January 2006. Available at: <http://coq.inria.fr/>.
11. The Cristal Team. Objective Caml, *version 3.09.1*. INRIA, January 2006. Available at: <http://caml.inria.fr/>.
12. The Focal Development Team. Focal, *version 0.3.1*. CNAM/INRIA/LIP6, May 2005. Available at: <http://focal.inria.fr/>.