Formal methods for exchange policy specification

Rémi Delmas & Thomas Polacsek*

Abstract

This paper introduces a modelling framework to perform automatic analyses on the specification of an information exchange policy. To avoid the increase of development costs and risks of uncontrolled dissemination of information, the specification errors need to be detected before the implementation phase. We propose a minimalistic core language to unambiguously represent an exchange policy specification and a gateway to logic solvers to verify some properties, namely: completeness, consistency, applicability and minimality. The aim is to check wether the formalisation of an exchange policy is consistent with user expectations.

1. Introduction

Today, it is clear that organisations are increasingly interconnected and exchange ever larger amounts of information. As an example, we can consider the extended enterprise, which is actually a group of firms sharing a common purpose, usually the manufacture and marketing of a product and which, through alliances, share resources and knowledge. In this context, information exchange occurs between the different companies, either through interconnected information systems, or federated platforms that serve as bridges between systems, or sometimes, even more trivially, by simple file sharing between agents. In addition, each organisation can belong to subsystems based on agreements, partnerships, subsidiaries belonging to the same group, etc.; depending on the subsystem, information exchange takes place according to different and potentially contradictory rules.

In this world of exchange, controlling the information flow appears essential. As a consequence, organisations need regulation norms to define rules that systems must follow for the information exchange. These rules ensure properties such as the fact that a particular agent is always informed of specific topics, the fact that no information can be released without prior consent of its owner, etc. A particular set of rules of this kind forms what we call an *exchange policy*.

Before considering the implementation of an information system, and all the problems that stem from the application of the exchange policy to the system, we need to define the notion of exchange policy itself and provide automatic analysis means to support the specification designer. The design errors that are not detected before the implementation phase of a policy increase development costs and risks of uncontrolled dissemination of information. In the more general field of specification engineering, recent works like [ABGR07], [CCR08] or [DDFPP11] propose to introduce formal methods in model-driven engineering processes, at early stages of model design and requirement formalisation. The proposed approaches mostly consist in using automatic constraint satisfaction techniques to perform various generic verification tasks on a given model and associated logical specification. Following the same line of thought, we propose to offer support to the designer of an exchange policy in order to verify that his formalisation correctly

*Authors version, Formal methods for exchange policy specification, 288-303, Advanced Information Systems Engineering - 25th International Conference, CAiSE 2013, Valencia, Spain, June 17-21, 2013. Proceedings. Lecture Notes in Computer Science 7908, Springer 2013, ISBN 978-3-642-38708-1

captures all aspects of the original problem without error, contradiction or redundancy, by offering a validation approach in the policy specification phase. In addition, a lightweight analyser based on state of the art pseudo-Boolean solving engines allows to analyse easily a policy when the specification of the system environment evolves, or when policy rules are added, removed or modified.

The goal of the proposed formal analyses is to support exchange policy specification. These analyses consist in establishing that the regulation of information exchange guarantees a set of properties that we call *generic*, namely: completeness, consistency, minimality and applicability. By "generic" we mean that these properties can be reasonably expected to hold for any policy specification, and somehow capture a simple form of *best practices* of policy engineering. As we will see, the completeness of a policy is the property that there are no situations not covered by the policy; the consistency of a policy is the property that rules do not contradict one another; the applicability of a policy is the property that it does not attempt to handle situations ruled out by domain constraints. Last, the minimality is the property that no rule of the policy can be deduced from the others.

The paper is structured as follows. In Section 2, we review existing logic representations of normative concepts. In Section 3, we give a definition of a formal modelling framework suitable to express exchange policies without ambiguities. In Section 5, we define the notion of completeness for a policy. In Section 6, we define the notion of consistency for a policy. Section 7 deals with notions of policy applicability and minimality. Last, Section 8 concludes the paper and outlines perspectives to this work.

2. Policies modelling

The notion of exchange policy is closely linked to the notion of norm. A norm in a multi-agent system is a set of statements that regulates the behavior of agents. It expresses which actions are mandatory, permitted or forbidden, for whom and under which conditions. According to the definition of *Information Technology Security Evaluation Criteria*, *"A System Security Policy specifies the set of laws, rules and practices that regulate how sensitive information and other resources are managed, protected and distributed within a specific system"*[ITS91]. This definition is very general, and many studies have attempted to formally define norms for information access, more precisely by defining formal frameworks for expressing rules on actions that enable an agent to access information, such as: reading a file, accessing a database [BC92][BC94][CBC08][CC97][JS92]. All these studies have in common the use of a modal operator to model the deontic normative aspects inherent in any policy.

In the literature, one also finds a large body of work in which various temporal logics are used to model the actual implementation of multi-agent systems. In this paper, we decide to focus only on the normative aspect of policies, adopting the notions provided by deontic logic.

Deontic logic is a formal framework dedicated to the expression of obligation and related concepts and to reasoning on them [Ce75]. To enable this, deontic logic defines the notion of *obligation* through a modal operator. A key property of this type of logic is that a proposition may be required to hold by the norms, without actually holding in the real world. If modal logic provides a formal framework, together with a semantics, which allows us to model and reason clearly and without ambiguity, it is not without limitations. One problem, raised by [McC97], is that even though modal logic helps to manipulate concepts and their associated rules, it is difficult to handle by ordinary people, and remains a tool limited only to logicians. One might however object to this argument that only natural language can be understood by everybody, bringing us back to all issues related to natural language ambiguities. We emphasise that one of the

problems of modal logic is the current limitations in tool support for evaluating the satisfiability of a formula. Since our goal is to provide automatic analyses to support the design and validation of an exchange policy, we need to be able to use efficient formal checking tools. At the present time, tools dedicated to modal logic are less efficient than standard logic solver tools [SV09].

In order to use plain first order logic rather than modal logic with a deontic modality, some works suggest to use a predicate to model the notion of obligation [ABB+03] [CGS06] [HW06]. In particular, [CM04] defines a model for *Role-Based Access Control* [FK92], named OrBAC (Organization Based Access Control). They introduce three predicates for obligation, permission and prohibition, which give abstract rights in an organisation, for a role, to perform an action for a view in a context. From there, they deduce what they call concrete permissions that model normative obligation, permission and prohibition, for a role to perform an action with an object. A simple example of this type of representation is the "prohibition for the role *secretary* to perform the action *modify file* on the object *medical record*".

Note that, in frameworks such as OrBAC, normative predicates have three places: subject, action and object. Their policies hence model actions represented by transitive verbs (divalent verb). This choice is motivated by the purpose of access control policies, which is to model the conditions under which agents can perform actions such as reading or writing data on various data sources. In our context, we are not interested in modelling actions in general, but rather in modelling actions of information exchange between two agents. Here, the information exchange actions correspond to ditransitive verbs (trivalent verbs), for instance: "to exchange", "to say", "to give" or "to send". In conclusion, if we follow the idea of representing deontic notions using predicates, it is not possible to represent an exchange policy using existing access control logical modelling frameworks.

Among all variants of first order logic, we chose *many sorted first-order logic* (MSFOL) [Gal87] over unsorted first-order logic as a background theory in which to model policies, simply because the use of sorts allows to express models with better compacity than in the unsorted case.

Of course, finding the right modelling methodology for multi-agent systems and defining the right ontologies for certain classes of systems is a research subject in its own right. At the present time, we have not yet addressed system modelling in such depth. The work presented here consists in providing a minimalist yet extensible modelling framework featuring automatic analysis supported by state of the art SAT/pseudo-Boolean solvers.

3. A Language for Exchange Policy Specification

The many-sorted first order logic is a good candidate general logical framework for modelling policies. Its signatures, syntax and semantics are described in the first subsection. The logical framework PEPS[1] is defined in the second subsection. PEPS syntax and semantics are directly inherited from MSFOL, but it offers a minimalist set of built-in sorts and predicates dedicated to policy modelling, and policy rules are logic formulas which must satisfy syntactical restrictions, which are also motivated and detailed in the second subsection. The third subsection introduces a example of specification.

3.1. Many-sorted First Order Logic

A many-sorted first order logic signature is defined as a quadruple

$$Sig = \{Sort, Var, Fun, \sigma\}$$

[1] PEPS is a recursive acronym for: Peps for Exchange Policy Specification.

where:

- *Sort* is a set of *sorts identifiers*, containing at least the sort *Bool*;
- *Var* is a set of *variables identifiers*;
- *Fun* is a set of *functions identifiers*;
- $\sigma : Fun \to Sort^* \times Sort$ associates a *prototype* with each function $f \in Fun$.

The prototype of a function f specifies the sort expected for each argument of f together with its return sort, and is written $((S_1, \ldots, S_N)\ S)$, where N is called the *arity* of f. If $N = 0$, f is called a *constant* of sort S. If the return sort is *Bool*, f is called a *predicate*. $Const \subseteq Fun$ is the subset of constants of the signature, and $Pred \subseteq Fun$ is the subset of predicates of the signature. A function identifier and its prototype declaration can be combined using the notation $f(S_1, \ldots, S_n) : S$, or even $f(a_1 : S_1, \ldots, a_n : S_n) : S$, whenever assigning names to function arguments is helpful for documentation purposes. The return sort is often omitted for predicates, usually declared as $p(a_1 : S_1, \ldots, a_n : S_n)$.

Well sorted terms and formulas are defined inductively as usual: a constant c of sort S is a term of sort S; assuming terms $t_1, \ldots, t_n$ of sorts $S_1, \ldots, S_n$ and a function $f(S_1, \ldots, S_n) : S$, then $f(t_1, \ldots, t_n)$ is a term of sort S. Terms of sort *Bool* are called *formulas*. If t_1, t_2 are terms of the same sort, then $t_1 = t_2$ is a formula. If p and q are formulas, then so are $\neg p$, $p \wedge q$, $p \vee q$, $p \implies q$ and $p \equiv q$ (representing logical negation, conjunction, disjunction, implication and equivalence, respectively).

Variable identifiers of the signature are used with quantifiers. A universally quantified formula (respectively existentially quantified formula) has the form $\forall x : S, p$ (respectively $\exists x : S, p$) where: x is a variable identifier said to be *bound* to the quantifier, p is a formula in which x is regarded as a term of sort S and must be *free* in p (i.e., not enclosed in the scope of any other quantifier found in p). The sort declaration for the variable can be omitted when it can be easily inferred by the reader from the use of the variable. A term or formula is *closed* when it does not contain any free variables, *i.e.* when all variables it contains are bound to some quantifier within the formula. For instance, $\exists x : s_1, f(x) \wedge f(y)$ is not closed because y is not bound and $\exists x : s_1, \forall y : s_2, f(x) \vee g(y)$ is a closed formula.

The semantics of many-sorted first order logic will not be detailed here, yet the usual truth conditions can be found in [Gal87]. Logical consequence through models is noted $\models$, *i.e.* we note $F \models G$ when each model of formula F is also a model of formula G and $\models F$ when F is a tautology.

3.2. Exchange Policies Specification

Now that the underlying logical framework is introduced, we can define the PEPS policy modelling framework with greater detail. A PEPS signature is essentially an MSFOL signature satisfying a few extra requirements. Since the goal of PEPS is to express what information items must, must not or can be exchanged, between whom and on what topic, we first assume that *at least* the following sorts are present in a PEPS signature:

- $\mathcal{A}$, which represents the agents of the system;
- $\mathcal{I}$, which represents information items;
- $\mathcal{T}$, which represents information topics.

In addition, the following *domain-predicates*, called *D-predicates* from now on, are assumed to be part of the signature:

- $Know(a : \mathcal{A}, i : \mathcal{I})$, which models the fact that an agent knows a piece of information;
- $Topic(i : \mathcal{I}, t : \mathcal{T})$, which expresses that a piece of information is relevant to a certain topic.

PEPS is not a locked-down language. It is possible to declare additional sorts and D-predicates in a PEPS signature besides those presented here. In this paper, we only give the core language to illustrate our automatic analyses. We will only use the D-predicates and sorts introduced so far in subsequent examples, but for realistic applications, the modelling framework can be easily extended with extra sorts and functions to capture finer details of domains and information exchange policies, such as: a predicate $Link(a_1 : \mathcal{A}, a_2 : \mathcal{A})$ to model that a communication link exists between agents a_1 and a_2; a sort $\mathcal{L}$ to model the different accreditation levels, a predicate $Ilevel(i : \mathcal{I}, l : \mathcal{L})$ to model that information item i requires accreditation level l for access, a predicate $Alevel(a : \mathcal{A}, l : \mathcal{L})$ to model that agent a has the accreditation level l, etc.

Well sorted PEPS terms, formulas and their semantics are defined just as in MSFOL.

The concept of obligation, more specifically the *obligation for an agent to send an information item to another agent*, must be readily available in the modelling framework. Unlike standard deontic logic, we will not have a generic obligation operator, but only the concept of *obligation to send information item i from agent a_1 to agent a_2*. We hence assume that three *normative-predicates*, called N-predicates from now on, are always present in a PEPS signature: $OSend(a_1 : \mathcal{A}, a_2 : \mathcal{A}, i : \mathcal{I})$, $PSend(a_1 : \mathcal{A}, a_2 : \mathcal{A}, i : \mathcal{I})$, $FSend(a_1 : \mathcal{A}, a_2 : \mathcal{A}, i : \mathcal{I})$, which represent respectively the obligation, permission and prohibition for an agent a_1 to send a information item i to another agent a_2.

A PEPS formula is called an *objective formula* if and only if it does not contain any *N-predicates*, in which case it only describes objective knowledge or facts about the domain rather than a normative requirement.

In standard deontic logic, the axiom (D) [Che80] expresses that if p is obligatory then p must be permitted. In PEPS we do not have obligation in the general sense but the obligation for some agent to send something to some other agent, so we translate this axiom to a logical property, that we call also (D).

Definition 1 (D)

$$\forall a_1, \forall a_2, \forall i, OSend(a_1, a_2, i) \implies PSend(a_1, a_2, i)$$

The property (D) expresses that if it is obligatory for an agent to send an information item to another agent, then it is also permitted for this agent to send this information item to this other agent. Next, we define the notion of *information exchange rule* in PEPS.

Definition 2 (Exchange rule)
An exchange rule is a closed PEPS *formula of one of the following syntactical forms:*

$$\forall x_1, \ldots, \forall x_n, \ (\phi \implies OSend(t_1, t_2, t_3))$$
$$\forall x_1, \ldots, \forall x_n, \ (\phi \implies PSend(t_1, t_2, t_3))$$
$$\forall x_1, \ldots, \forall x_n, \ (\phi \implies FSend(t_1, t_2, t_3))$$

where:

- *$x_1, \ldots, x_n$ are all the variables identifiers occurring in ϕ, t_1, t_2 and t_3;*
- *ϕ is a quantifier-free* objective formula*;*

- t_1, t_2 *are quantifier-free terms of sort* $\mathcal{A}$*;*
- t_3 *is a quantifier-free term of sort* $\mathcal{I}$*.*

In definition 2, the formula ϕ is without quantifiers and without normative predicates, but besides these restrictions all operators, constants, functions and variables (necessarily bound to one of the rule's quantifiers) are allowed. Moreover, t_1, t_2 and t_3 are not necessarily closed, so they can contain constants, functions and *variables*. Of course, the rule formula being closed, the variables in these terms are necessarily bound to one of the rule's quantifiers.

Each rule expresses under what condition it is obligatory, permitted or forbidden for an agent to send a piece of information to another agent. For instance, given constants $i_1, i_2, i_3 : \mathcal{I}$ and constant $b : \mathcal{A}$, the formula $\forall a, (Know(a, i_1) \vee Know(a, i_2)) \implies OSend(a, b, i_3)$ is a correct exchange rule, but the formula $\forall a_1, \forall a_2, \forall i, \top \implies \neg OSend(a_1, a_2, i)$ is not correct, because of the negation on the right hand side of the implication.

Definition 3 (Exchange policy)
An exchange policy EP is a collection of exchange rule formulas.

Definition 4 (Exchange policy specification)
An Exchange Policy Specification EPS is a pair $\langle \Sigma, EP \rangle$ *where:*

- Σ *is a satisfiable set of objective formulas describing facts and knowledge about the domain;*
- *EP is an exchange policy as described in Definition 3;*

3.3. Example

Consider this simple example: an Earth Observation System composed by several observing systems sharing environmental data, for the purpose of disaster prevention and management (this example is strongly inspired by GEOSS: Global Earth Observation System of Systems). Each observing system is owned by a nation and exchanges observations with others in order to build a global picture of the Earth's condition. Because the system is civilian and is open to all nations, and because some of the observing systems are military and some are civilian, nations do not want any information with potential military utility to get disseminated through this system. Moreover, in the context of climate disaster awareness, this system has a distinguished agent, the *Tsunami Warning Center* or *TWC*.

This system has rules which specify in what context, under what conditions, who has the obligation, prohibition or permission to release any information, to whom. For the moment, let us assume that the policy consists of only two rules: (a) *"as part of the tsunami prevention, all information about the topic tsunami (such as a submarine landslide) must be sent to the Tsunami Warning Center"*; (b) *"the system should not disclose information relating to military topics"*.

These two rules are expressed in the PEPS framework as follows (with constants $TWC : \mathcal{A}$, $tsunami : \mathcal{T}$ and $mili : \mathcal{T}$):

$$\text{(a)}\ \ \forall a, \forall i,\ \ Topic(i, tsunami) \wedge Know(a, i) \implies OSend(a, TWC, i)$$
$$\text{(b)}\ \ \forall a_1, \forall a_2, \forall i,\ \ Topic(i, mili) \wedge Know(a_1, i) \implies FSend(a_1, a_2, i)$$

Note that PEPS provides a minimalist and extensible modelling language with only three sorts but, even if specifying topic taxonomies is not the issue discussed here, it is possible to express simple notions like a *parent* relationship among topics using the logical implication and extra constraints. In our example, if *military* is the parent topic of *troop*, we would write

$Topic(i, troop) \implies Topic(i, mili)$ which means all information about the topic *troop* is also about the topic *military*. From this and rule (*b*), we can easily deduce: $\forall a_1, \forall a_2, \forall i, Topic(i, troop) \wedge Know(a_1, i) \implies FSend(a_1, a_2, i)$, the action of sending information about the topic *troop* is prohibited.

4. Automated Model Finding Techniques

4.1. Bounded model checking

Our aim is to provide automatic tool support to users defining exchange policy specifications. Analyses should be simple to configure and to use and as automatic as possible. In order to do so, we harness the power of automated model finding techniques for logical constraints.

In approaches such as [ABGR07], [CCR08] or [DDFPP11], the addressed verification problems are formalised in (many-sorted) first order logic and are hence undecidable in the general case, *i.e.* it is not possible to determine if an arbitrary MSFOL (or PEPS, by extension) formula is valid in finite time. So, in the mentioned approaches, authors used a *bounded model checking* approach: for an analysis, instances which satisfy the formulas under analysis are searched for in a bounded universe and the cardinality is increased by the user from iteration to iteration. Even though each bounded instance can be arbitrarily large, it remains decidable. A key feature of model finding approaches is that, when a required property does not hold on a specification, a counter example is produced, which helps the user understand the problem and fix the specification.

4.2. The PEPS implementation

Because of the decidability issue, we also adopt a bounded model checking approach to analyse PEPS specifications. By doing so, we lose the ability to prove genuine first order theorems about PEPS specifications. Indeed, the proposed analyses are only valid on systems up to a certain number of agents, information items, topics, etc. But on the other hand, we obtain a higher degree of automation and simplicity for the user. Our analyser is just a black box to which logical queries about the policy specification can be addressed with minimal parameter configuration.

The analyses detailed in the next sections are implemented using our own grounding tool for MSFOL formulas [DDFPP11], which interprets sorts over finite discrete domains.

With our tool, the only way to conduct an analysis is by transforming a given MSFOL formula into an equivalent pseudo-Boolean logic formula and by checking its satisfiability using a compatible solver. For instance, given two formulas F and G, if the goal is to verify that $F \models G$, then $F \wedge \neg G$ is translated and checked for unsatisfiability. Whenever the pseudo-Boolean solver returns a model of the given formula, the tool translates it back into the PEPS specification language to help debug the specification.

The pseudo-Boolean satisfiability problem can be seen as an extension of the SAT problem. A pseudo-Boolean instance is a conjunction of constraints of the form $\Sigma_i a_i\ l_i \leq k$, where a_i and k are integer coefficients, and l_i a literal (a boolean variable or its negation). Today's pseudo-Boolean solvers use a standard file format named OPB and periodically the international PB-Eval competition[2] allows to benchmark the latest evolutions of the field.

The main ideas behind the grounding and propositional encoding procedure we implemented follow. First, the grounding operation works by representing sorts using finite sets of constants, introducing as many fresh constants[3] as needed by the sorts cardinalities. For instance, the sort S

[2] `http://www.cril.univ-artois.fr/PB09/`

[3] constants which name is not already used elsewhere in the specification

of cardinality n is represented by the set of fresh constants $\{S_1, \ldots, S_n\}$. Then, domain constants are substituted for variables and propagated in all quantified expressions. Universal quantifiers are expanded as conjunctions and existential quantifiers as disjunctions. After this step, there are no quantifiers or variables left in the challenge formula, only constants, predicate and function applications and logical operators.

Second, the grounded formula is transformed into a propositional formula using a *bitvector* encoding (bitvectors are vectors of propositional variables): identifiers for constants of given sorts are encoded using bitvectors of appropriate size; predicates are encoded using a propositional variable for each point of their domain of definition; functions returning a given sort are encoded using a bitvector of appropriate size for each point of their domain of definition; function and predicate applications are encoded using advanced bitvector constructs.

Last, bitvector expressions are transformed into clauses (in pseudo-Boolean form) using standard Tseitin rules [Tse68]. Obviously, only clausal encodings are generated by following this approach, but we plan on allowing cardinality constraints and generic pseudo-Boolean constraints in the PEPS language in the future, hence the use of pseudo-Boolean logic as target language. After this translation we obtain a pseudo-Boolean formula that can be analysed using any solver supporting the OPB format. One important feature to notice about the tool is that each translation step is cached and reversible, meaning the tool is able to translate a model returned by a solver back to the PEPS level. For solving, we favor Sat4j [BP10], yet alternatives are available, like WBO [MSP09] or Minisat+ [ES06]. Most of these solvers are open source.

5. Completeness of a Policy

5.1. Definition

A policy specification is not *complete* whenever there are situations in which agent behavior is not explicitly constrained by the policy, that is to say, when at least one *non liquet*[4] exists. This notion of completeness shall not be mistaken for that of completeness for a formal system such as MSFOL. On the one hand, if the policy says nothing about the dissemination of a piece information, an agent could choose a permissive approach[5], everything that is not specified is allowed, and in this case the agent could send sensitive information. On the other hand, if the agent chooses a prohibitive approach, in which everything that is not specified is prohibited, then communication in the system could be forbidden. The incompleteness of a policy may hence become a major issue if it relates to sensitive systems where a *non liquet* gives the possibility for agents to act with important consequences.

Here, we give a simple formal definition for policy specification completeness: a policy specification $EPS = \langle \Sigma, EP \rangle$ is complete if and only if, for any agent who knows a piece of information, the policy rules state whether or not to send it to any another agent, in any situation allowed by the domain model Σ. In other words, the policy specification is complete if it is possible to deduce from Σ and EP that each agent, for any piece of information, in any situation, is either allowed, obligated or forbidden to send it to any other agent.

Definition 5 (Completeness of a policy specification)
Let $EPS = \langle \Sigma, EP \rangle$ *be an exchange policy specification, EPS is* complete *if and only if*

[4] In law, a situation where there is no applicable law is a *non liquet*.
[5] The legal principle *nulla poena sine lege* "no penalty without a law".

$$(\bigwedge_{r \in EP} r) \wedge \Sigma \wedge D \models \forall a_1, \forall a_2, \forall i,\ Know(a_1, i) \implies (OSend(a_1, a_2, i) \vee PSend(a_1, a_2, i) \vee FSend(a_1, a_2, i))$$

Please note that a policy EP on its own could be incomplete and only become complete once conjoined with domain constraints Σ, which can precisely rule out *non liquets*.

5.2. Example

In the Earth Observation System example (setion 3.3), analysing the policy specification $\langle \emptyset, \{a, b\} \rangle$ with the PEPS analyser reports that it is not complete. A simple counter example is returned, in which information is linked to no topic, and hence does not fall under any of the policy rules.

A possible solution to this problem is, for example, to suppose that in the system, there is no information without topic. This should be expressed not in the policy, but rather in the domain constraints. Therefore, we add the following formula (r) (for relevancy) which says *"any information is relevant to at least one topic"*:

$$\text{(r)}\ \ \forall i,\ \ \exists t, Topic(i, t)$$

Despite this addition, $\langle \{r\}, \{a, b\} \rangle$ is still not complete. Indeed, the analyser returns a new counter example in which information is *tsunami* related. It reveals that the policy has no rules specifying what an agent is supposed to do with information about the topic *tsunami* when dealing with an agent that is not the *TWC* agent. We extend the policy with the rule (c): *"it is permitted to exchange information on the topic tsunami"*, which is formalised as follows:

$$\text{(c)}\ \ \forall a_1, \forall a_2, \forall i,\ \ Topic(i, tsunami) \wedge Know(a_1, i) \implies PSend(a_1, a_2, i)$$

At this point a remark is necessary. As already mentioned, our analyser only proves the existence or absence of a model in a finite universe, which cardinality is specified by the user. In our example, if we choose the cardinality of the sort $\mathcal{T}$ to be equal to two, then the policy specification appears complete. Indeed, "all possible topics" is reduced to two topics *tsunami* and *mili*, and the policy does cover all possible cases assuming only these two topics. But, if more than two topics are allowed, our policy specification no longer appears complete. In this case, the analyser returns a model in which information is relevant neither to the topic *tsunami*, nor to the topic *mili* and for which no rule applies. Choosing sort cardinalities when analysing the model is left to the user.

To obtain completeness for the example policy, we must add a rule (d) which is: *"there is no particular constraint on the exchange of information not relevant to the military topic"*. This rule corresponds to the following formula:

$$\text{(d)}\ \ \forall a_1, \forall a_2, \forall i,\ \ \neg Topic(i, mili) \wedge Know(a_1, i) \implies PSend(a_1, a_2, i)$$

Now, we can check with our analyser that the specification $\langle \{r\}, \{a, b, c, d\} \rangle$ is complete. If we increase sort cardinalities, for example to 10 agents, 10 topics and 10 pieces of information, the specification is still complete. Of course, we do not prove the completeness of the specification for an infinite universe. Nevertheless, we are more confident after analysing the specification than without any analysis, and the analyser has been of great help in the task of writing and fine tuning the specification.

6. Consistency of a Policy

6.1. Definition

In logic, a theory is the collection of all truths that can be derived from a core set of assumptions. A theory is consistent if it does not contain a contradiction. In the context of confidentiality policies, in [BC94] two policies are consistent when conjoined if no user can have the permission, according to the first policy, to know something and the prohibition, according to the second, to know it. For us, an exchange policy is not consistent when some situation exists in which it is both obligatory and forbidden, or permitted and forbidden, for an agent to send a piece of information to another agent. The policy inconsistency is an important issue because both authorising and prohibiting a same behavior forbids the application of the policy as a whole. The Algorithm 1 yields, given a policy, a set of situations exhibiting inconsistency between pairs of rules, shall any such situation exist. For each possible pair of rules having conflicting normative predicates as conclusions, the algorithm generates a formula which, if satisfiable, indicates a situation in which the premises of both rules are satisfied, and in which the terms used as arguments of the conflicting predicates unify, hence showing that conflicting normative requirements must be applied to the same agents and same information item. Not mentioned in this algorithm is the implicit variable renaming performed on elements taken from r' to avoid variable clashes when forming the satisfiability objective.

Definition 6 (Consistency of a policy specification)
Let $EPS = \langle \Sigma, EP \rangle$ be an exchange policy specification, EPS is consistent *if and only if the result of Search for Non-Consistent rules $SNC(\Sigma, EP)$ is empty.*

Algorithm 1 Search for Non-Consistent rules (SNC)

Require: $\langle \Sigma, EP \rangle$, an exchange policy specification
Ensure: S, a set of tuples (r, r', m) with r, r' exchange rules, m a model.

```
S ← ∅
for all r ∈ EP do
    if r is of the form ∀x_1, ..., ∀x_n, (φ ⟹ FSend(t_1, t_2, t_3)) then
        for all r' ∈ EP do
            if (r' is of the form ∀x'_1, ..., ∀x'_m, (φ' ⟹ OSend(t'_1, t'_2, t'_3))
                                or ∀x'_1, ..., ∀x'_m, (φ' ⟹ PSend(t'_1, t'_2, t'_3))) then

                if (m is a model of ∃x_1, ..., x_n, x'_1, ..., x'_m,
                        φ ∧ φ' ∧ Σ ∧ (t_1 = t'_1) ∧ (t_2 = t'_2) ∧ (t_3 = t'_3)) then
                    S ← S ∪ (r, r', m)
                end if
            end if
        end for
    end if
end for
```

6.2. Example

In our example, the exchange policy $\langle \{r\}, \{a, b, c, d\} \rangle$ is not consistent. Indeed, the analyser discovers two contradictions, one between rules (a) and (b) and another between rules (b) and (c).

For the first contradiction, a counter example is returned, in which an information item is relevant of to two topics, both *tsunami* and *mili*, and the agent who receives the information is *TWC*. A very naive solution is to suppose that each information item is relevant of one and only one topic. More realistic, with PEPS, we could also consider that information related with *mili* has an accreditation level M, it is permitted to exchange this kind of information with agents whose accreditation level is M, and that *TWC* is accredited at level M. To keep things simple, we decided to replace the rule (b) with (b') *"an agent should not disclose information related to mili topics to an agent other than TWC"*:

$$\text{(b')}\ \ \forall a_1, \forall a_2, \forall i,\ \ Topic(i, tsunami) \wedge Topic(i, mili) \wedge Know(a_1, i) \wedge (a_2 \neq TWC) \implies FSend(a_1, a_2, i)$$

In the same way, we replace the rule (c) with (c') *"an agent should not disclose information related to mili topics but not related to tsunami topics"*:

$$\text{(c')}\ \ \forall a_1, \forall a_2, \forall i,\ \ Topic(i, tsunami) \wedge \neg Topic(i, mili) \wedge Know(a_1, i) \implies PSend(a_1, a_2, i)$$

Yet, with this modification, the analyser informs us that the specification $\langle \{r\}, \{a, b', c', d\} \rangle$ is not complete. To obtain completeness we must add one more rule (e)*"military information not related to tsunami should not be sent between agents"*:

$$\text{(e)}\ \ \forall a_1, \forall a_2, \forall i,\ \ Topic(i, mili) \wedge \neg Topic(i, tsunami) \wedge Know(a_1, i) \implies FSend(a_1, a_2, i)$$

7. Other Properties of a Policy

In this section, we define two new properties for policies, aiming at detecting cases of over-specification: *applicability* and *minimality*. An applicable and minimal policy is such that each rule can be applied in at least one situation allowed by domain constraints, and contains no redundant rule with respect to others.

7.1. Definitions

A policy is *applicable* under domain constraints Σ if and only if for each of its rules, there exists a situation in which the rule applies. The underlying idea is that in an applicable policy, there are no rules which premises are contradicted by the domain constraints.

Definition 7 (Applicability of a rule)
Let Σ be a set of domain constraints and r an exchange rule as defined in Definition 2, section 3, i.e. *$r = \forall x_1 \ldots \forall x_m(\phi \rightarrow \psi)$. r is applicable relatively to Σ if and only if*

$$\Sigma \wedge D \models \exists x_1 \ldots \exists x_m \phi$$

Definition 8 (Applicability of a policy)
Let $EPS = \langle \Sigma, EP \rangle$ be an exchange policy specification, EPS is applicable if and only if for each rule r of EP, r is applicable relatively to Σ.

So, the applicability of a rule is defined as a satisfiability problem for each rule's ϕ component under assumptions $\Sigma \wedge D$. The unsatisfiability of the rule's premises under environment constraints reveals the non-applicability of the rule, *i.e.* proves that no situation can satisfy the premises.

Another interesting notion for a policy is that of minimality. A policy is said to be *minimal* if none of its rules can be deduced from the others under the domain constraints.

Definition 9 (Minimality of a policy)
Let $EPS = \langle \Sigma, EP \rangle$ be an exchange policy specification with $EP = \{r_1, \ldots, r_n\}$. EPS is minimal if and only if there is no i such that:

$$\Sigma \wedge D \models (\bigwedge_{k \in [1,n], k \neq i} r_k) \implies r_i$$

In the same way as for applicability, showing the minimality of a policy is achieved by solving a series of satisfiability problems, each exhibiting a situation in which a certain rule is the only rule of the policy whose premises are satisfied, hence showing its independence with respect to the other rules.

7.2. Example

Coming back to our example, the analyser allows to prove that all rules are applicable: the policy specification $\langle \{r\}, \{a, b', c', d, e\} \rangle$ is applicable.

However, the analyser reports that the specification is not minimal. Indeed, it can be proved that the rule (c') can be deduced from the others. If we choose to reduce the policy specification by removing the rule (c'), we can successfully check with the analyser that the specification is still complete and consistent.

Finally, the exchange policy specification $\langle \{r\}, \{a, b', d, e\} \rangle$ is complete, consistent, applicable and minimal.

8. Conclusion

In this paper we proposed a logical modelling framework suitable to express information exchange policies. We also defined essential generic properties such as completenes, consistency, applicability and minimality, and provided procedures for checking these properties using currently available logic solvers. The proposed framework is extensible, as it is possible for the user to declare any additional sort, function, or predicate needed for modelling, such as the roles played by an agent in an organisation, or the accreditation level of information or agents, etc.

The bounded model checking approach and the pseudo-Boolean solvers back-ends used in the analyses proposed in this paper have already passed with great success scalability tests on real world instances in many different application domains like software verification, hardware verfication etc.Therefore, we are confident in their performance for policy verification.

Our future work will cover two complementary aspects of policy verification and specification. First, on the technical side of formal verification, we will study if and how the latest features of logic solvers can be used to help the verification of policies. In particular minimal unsat core generation may allow to discover the details of why a rule is unapplicable under a given set of environment constraints. Concerning the completeness of an exchange policy, future work could focus on the possibility to support the specification. Automatic model enumeration features will be studied in relation with the abduction reasoning principle to try to automatically extend an incomplete policy and make it complete.

Second, we intend to apply the proposed approach to model a real-world information exchange policy in the domain of space situation awareness. The targeted application being somewhat complex, we will investigate and develop the methodology of domain modelling and policy

specification further. Dealing with information systems of this kind already brings new questions to mind, such that of defining and studying other generic properties of policies. For example, we might be interesed to verify that vital information can always reach the relevant agents for appropriate action. For each new generic property, we shall proceed in the same way and give it a formal definition in our modelling framework, and define associated verification procedures. Also, supporting a variant of MSFOL with subsorting could allow for a better and more concise way of modelling of complex application domains. Subsorting indeed allows to model hierarchies of concepts or entities with greater ease. We also plan on allowing the specification and verification of user-specified properties, in addition to the four generic properties discussed in this paper.

References

[ABB+03] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03)*, June 2003.

[ABGR07] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. UML2Alloy: A Challenging Model Transformation. In G. Engels, B. Opdyke, D.C. Schmidt, and F. Weil, editors, *ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems*, volume 4735 of *LNCS*, pages 436–450, Nashville, USA, 2007. Springer.

[BC92] Pierre Bieber and Frédéric Cuppens. A logical view of secure dependencies. *Journal of Computer Security*, 1(1):99–130, 1992.

[BC94] Pierre Bieber and Frédéric Cuppens. *Expression of confidentiality policies with deontic logic*, pages 103–123. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[BP10] Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *JSAT*, 7(2-3):59–6, 2010.

[CBC08] Nora Cuppens-Boulahia and Frédéric Cuppens. Specifying intrusion detection and reaction policies: An application of deontic logic. In *DEON*, pages 65–80, 2008.

[CC97] L. Cholvy and F. Cuppens. Analyzing consistency of security policies. In *Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on*, pages 103–112, may 1997.

[CCR08] Jordi Cabot, Robert Clarisó, and Daniel Riera. Verification of uml/ocl class diagrams using constraint programming. In *ICSTW '08: Proceedings of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop*, pages 73–80, Washington, DC, USA, 2008. IEEE Computer Society.

[Ce75] H. N. Castanẽda. *Thinking and doing*. D. Reidel, Dordrecht, 1975.

[CGS06] Laurence Cholvy, Christophe Garion, and Claire Saurel. Information sharing policies for coalition systems. In *NATO RTO-IST-062 Symposium on Dynamic communications management*, 2006.

[Che80] B. F. Chellas. *Modal logic, an introduction*. Cambridge University Press, 1980.

[CM04] Frédéric Cuppens and Alexandre Miège. Administration Model for Or-BAC. In *Computer Systems Science and Engineering (CSSE'04)*, volume 19, 2004.

[DDFPP11] Rémi Delmas, David Doose, Anthony Fernandes Pires, and Thomas Polacsek. Supporting model-based design. In *MEDI'11*, Obidos, Portugal, 2011.

[ES06] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into sat. *JSAT*, 2(1-4):1–26, 2006.

[FK92] David Ferraiolo and Richard Kuhn. Role-based access control. In *In 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[Gal87] Jean H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*, chapter 10, pages 448–476. Wiley, 1987.

[HW06] Joseph Y. Halpern and Vicky Weissman. Using first-order logic to reason about policies. *CoRR*, abs/cs/0601034, 2006.

[ITS91] ITSEC. Information technology security evaluation criteria (itsec): Preliminary harmonised criteria. Technical Report Document COM(90) 314, Version 1.2., Commission of the European Communities, 1991.

[JS92] Andrew J. I. Jones and Marek J. Sergot. Formal specification of security requirements using the theory of normative positions. In *Proceedings of the Second European Symposium on Research in Computer Security*, ESORICS '92, pages 103–121. Springer-Verlag, 1992.

[McC97] John McCarthy. Modality, si! modal logic, no! *Studia Logica*, 59(1):29–32, 1997.

[MSP09] Vasco M. Manquinho, João P. Marques Silva, and Jordi Planes. Algorithms for weighted boolean optimization. In Oliver Kullmann, editor, *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 495–508. Springer, 2009.

[SV09] Roberto Sebastiani and Michele Vescovi. Automated reasoning in modal and description logics via sat encoding: the case study of k(m)/alc-satisfiability. *J. Artif. Intell. Res. (JAIR)*, 35:343–389, 2009.

[Tse68] G. S. Tseitin. On the complexity of derivations in the propositional calculus. *Studies in Mathematics and Mathematical Logic*, II, 1968.