Verification of SGAC Access Control Policies using Alloy and ProB

Nghi Huynh*†, Marc Frappier†, Amel Mammar‡ and Régine Laleau*
†Université de Sherbrooke, Québec, Canada
*Université Paris-Est Créteil Val de Marne, France
‡Institut Mines-Télécom/Télécom SudParis, France

***Abstract*—This paper investigates the verification of access control policies for SGAC, a new healthcare access-control model, using Alloy and ProB, two first-order logic model checkers based on distinct technologies. SGAC supports permission and prohibition, rule inheritance among subjects and resources and conflicts resolution. In order to protect patient privacy while ensuring effective caregiving in safety-critical situations, we check different properties such as accessibility, ineffective rule detection. Our performance results show that ProB performs two orders of magnitude better than Alloy. Results are promising enough to consider ProB for verifying patient policies in SGAC.**

***Index Terms*—healthcare, access control, consent management, formal model, verification, Alloy, ProB.**

I. Introduction

With medical data being stored electronically, access control over these sensitive data has become crucial and compulsory. But control over medical data is not an easy task. Access control must ensure the patient's privacy, not hinder health worker's work and not endanger the patient's life. SGAC (*Solution de Gestion Automatisée du Consentement* — Automated Consent Management Solution) [12] is an access control method which has been developed for the Sherbrooke University Hospital. It allows patients and the hospital to specify fine-grained access control rules over the medical data. In order to ensure patient safety and privacy, properties must be checked on the access control policies. For instance, the hospital would like to ensure that crucial patient data is available when the patient's life is in danger. Patients want to ensure that sensitive data that could damage their reputation, employability or social relationships are only disclosed to the appropriate persons in the right context. The flexibility of SGAC's policy language makes it mandatory to use automated verification techniques to check properties of SGAC access control policies. Reusing existing model-checking tools is more cost effective and less risky in terms of long-term maintenance, while allowing for leveraging of future improvements.

There are three main classes of model checkers for first-order logic: SAT-based approaches like Alloy [13], constraint-based approaches like ProB [14], and SMT-based approaches like CVC4 [5] and Z3 [4]. ProB and Alloy are easier to use to model SGAC policies and they have both been shown to be useful in solving graph problems and complex first-order constraints like the ones used in SGAC [7], [8].

In this paper, we evaluate the applicability of Alloy and ProB for checking properties of SGAC policies such as verifying that a user has access to a document in a given context, or identify the rules that are overridden by other rules, and thus have no effect on the access granting decisions, and hence may denote misrepresented safety/privacy requirements.

The rest of this paper is structured as follows. Section II introduces SGAC, and its conflict resolution mechanism for ordering rules applicable to a given request. Section III presents Alloy and ProB, and a brief overview of the formalisation of SGAC in these respectives languages. The complete specifications are available at [10]. Verification performance results are given in Section IV. Section V compares our findings with similar work on access control. We conclude this paper with an appraisal of our work in Section VI.

II. SGAC: presentation

We present here all elements of SGAC needed to understand property verification. A more detailed description can be found in [12]. SGAC handles requests made by users to access documents and returns a permission or a prohibition depending on the rules specified by the patients, the hospitals or required by laws and regulations. To evaluate requests, SGAC relies on two directed acyclic graphs (DAG): the subject and the resource graphs. The subject graph represents the hierarchy which mirrors the functional organisation chart or any grouping of users relevant for access control. The resource graph represents the taxonomy of medical documents and their organisation in the healthcare facilities. A request is made by a single user (sink of the subject graph) to do an action on a single document (sink of the resource graph). Depending on its *modality*, a rule authorises or denies a *subject* (node of the subject graph) to do an *action* on a *resource* (node of a resource graph) under a *condition*.

A. Behaviour and conflict resolution

A rule applies to a request when i) there is a path from the rule's subject to the request's subject, ii) there is a path from the rule's resource to the request's document, and the rule's condition holds. When many rules with different modalities apply to the same request, a situation, typically called a *conflict* in the literature, arises. To decide whether access is granted or denied, we define an ordering on rules. Rules are assigned priorities to help the ordering. The rule with the "highest" precedence determines the access decision. Let r_1, r_2 be two applicable rules for a request.

1) If r_1 has a *smaller* priority than r_2, we say that r_1 has precedence over r_2.
2) If r_1 and r_2 have the same priority, and if the subject of r_1 is more specific than the subject of r_2 (*i.e.*, the subject of r_1 is a descendant of the subject of r_2 in the subject graph), then r_1 has precedence over r_2.
3) If r_1 and r_2 have the same priority, and neither of their subjects is more specific than the other, then prohibitions have precedence over permissions.

This ordering is not total. There may be two rules r_1, r_2 such that neither of them precedes the other. However, in such a case, r_1 and r_2 have the same modality, thus there is no conflict and the decision is the modality of these elements with highest precedence in this ordering. The conflict resolution method relies on the fact that, generally, a rule which targets a smaller group (inclusion-wise) than other rules should have precedence over these. This conflict resolution method is absolutely autonomous and does not require the intervention of an external actor.

B. Example

We illustrate the presented behaviour with the following example: let Bill be an anaesthetist and a surgeon. Since he has two profiles, he inherits access privileges from both of them. In Fig. 1, rules which apply to a request of Bill to read the document D are: r_1, r_2, r_3 and r_4. We suppose they share the same priority. In the case where all these four rules are active under the same context (*i.e.*, their condition holds): surgeons are not able to access the document D while anaesthetists can, making Bill unable to access the document. We have r_1 that is less specific than r_2, and the same goes for r_3 and r_4. Since r_2 and r_4's subjects are incomparable, precedence is given to the prohibition, resulting in a prohibition for Bill's request. If there is a context where only r_1, r_3 and r_4 are active, Bill's request would be granted in this context since anaesthetists would be allowed to access the document.

III. Formalisation of SGAC

In this section, we present an overview of the formalisation of SGAC in Alloy and B. For the sake of concision, most of the specification is omitted. The complete specifications are available at [10], together with an expanded version of this paper highlighting the key elements of the specifications.

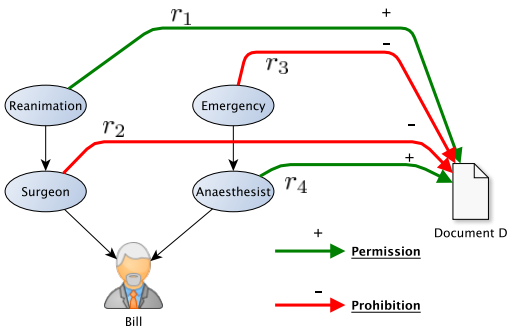


Fig. 1: Example of rule precedence

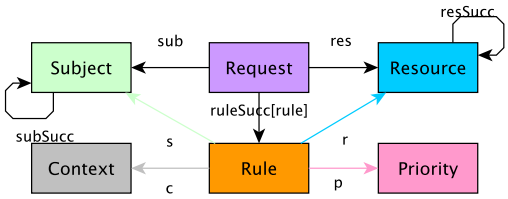


Fig. 2: Basic structure of SGAC

A. Formalisation of SGAC in Alloy

Alloy is a formal language for describing relational structures. Relations are declared using an object-oriented syntax. Alloy is supported by a tool, the Alloy Analyzer, for analysing and exploring the relational specification. It is a first-order logic model finder: the solver takes the constraints of a specification and finds instances that satisfy them. Alloy offers a customisable graphical interface and an evaluator which improves the user experience and greatly helps in understanding the model and counterexamples. Its graphical interface is particularly convenient when handling complex graphs.

The data structures called signatures needed to model SGAC are presented in Figure 2, which also illustrates the relationships between these signatures. The relations `subSucc`, `resSucc` and `ruleSucc` denote the edges of the subject, the resource and rule graphs. A rule graph denotes the precedence among the rules, and must be created for each request and context since the ordering is made on rules applicable to the request in a given context. However to reduce the computational burden, we came with optimisations: instead of computing a rule graph for each request and context, we only compute a graph for each context. Each rule graph determines the result of the request given a context: instead of looking for sinks, we look for vertices that have all their succesors' conditions unsatisfied in the given context that we call *pseudo-sinks*. The crucial part of the Alloy formalisation lies within the

definition of the rule ordering `ruleSucc` defined over rules that apply to a given request: i) rules are compared by their priority and subject specificity; ii) only maximal elements (thus of the same priority) are compared by their modality.

B. Formalisation of SGAC in B

The B language is used to specify systems, by describing state variables and operations that modify these state variables. A B specification consists of B machines. A machine contains the following clauses: sets, constants, properties of the constants, state variables, invariant of the state variables, initialisation and operations. ProB [14] is a model checker and animator for the B language [1]. It uses first-order logic, arithmetic, sets, relations and functions.

In order to model SGAC in B, we use the model finder and constraint solving features of ProB. We tried two approaches. In the first approach, we model the SGAC policies and the properties using solely sets, constants and properties clause of a B machine, in a way pretty similar to the Alloy specification. This approach failed because ProB was unable to solve the constraints in an efficient manner. In the second approach, we use variables and operations to impose an order in which the constraints can be efficiently solved by ProB. Thus, some of the data is represented as constants and properties, and the others are represented as state variables which are computed in a specific order in the initialisation clause of the machine, and operations are used to compute values of the properties to check. This approach is highly successful and represents a decisive advantage for ProB in comparison with Alloy. Our final model is structured as follows:

1) Declaration of the subject and resource graphs and the rule base in the constants and properties clauses.
2) Declaration of the rule ordering structures as variables.
3) Initialisation of the variables with a sequence of instructions, with regards to the dependency between variables.
4) Declaration of the operations which represent the different properties we want to verify.

IV. Performance tests

To test our models, we randomly generate graphs and requests. We control the following parameters: the number of vertices in the graphs, the number of contexts, the number of rules and the number of requests. We check all four properties by varying only one parameter at a time. For a given value of the parameters, we generate at least 6 models and compute the average execution time for checking the properties of the models. The properties we verify are:

- access: a user can effectively have access to a document under a given context;
- hidden data: there is no data completely hidden from all health workers;
- granting contexts: detection of the contexts which grant access given a request;
- ineffective rules: detection of the rules that never affect the result of a request.

For Alloy, each property is verified by running a check or run command. For ProB, one execution can verify all four properties at the same time. Tests were performed on a virtual server (Intel Xeon 3.10GHz) using Java 1.7 with 12GB of RAM. The results presented in Fig 3 show the outstanding performances of ProB and that the solving time is constant with regards to the number of contexts, linear with the number of vertices and exponential with the number of rules.

V. Related work

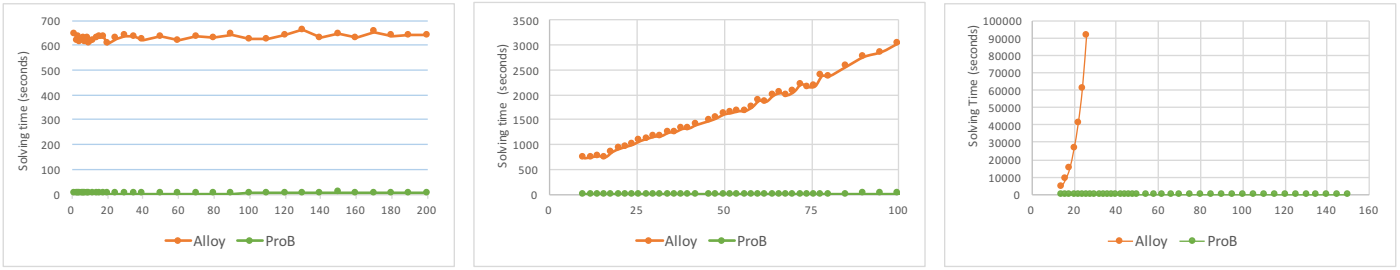
RBAC [21], a standardized and well-known access-control model based on roles, allows for specification of permissions associated to a role, to execute actions on resources. The lack of prohibitions and conditions in this model makes the verification of an access property easier, since there are no conflicts. However, it has been shown in [17] that RBAC is inappropriate for fine-grained access control as found in healthcare requirements like CIUSSS-Estrie's (*Centre Intégré Universitaire de Santé et de Services Sociaux* or Integrated University Health and Social Services Center). RBAC has been formalised in Z [18], [19] and B [11] where traditional RBAC properties are checked (role activation, role hierarchy and separation of duty).

OrBAC [6], a logic-based access-control model, introduces the notion of organisation and includes among other things explicit prohibitions and contexts. A rule can be defined to be only applicable in specific contexts. Conflicts are detectable by static checking with the Prolog-based tool MotOrBAC [3]. OrBAC does not fit our needs since it does not have an automatised conflict resolution, and once a static check reveals a conflict, a human intervention is required to solve it.

XACML [20] is an attribute-based access-control language that features prohibitions and conditions, and allows to determine how conflicts are managed by rule combination algorithms. As shown in [17], XACML does not natively support rule inheritance, since it does not include a graph of subjects or resources; it can be simulated using paths in resource name, but this complexifies the maintenance of a rule base, while providing poor performance for very large rule bases. XACML has been formalised in several ways ([2], [9], [16]). These formalisations allow for access property verification but cannot be reused easily with our rule ordering.

VI. Conclusion

We have presented an approach to verify four types of crucial properties for SGAC access control policies using Alloy and B. B performs significantly better (at least two orders of magnitude) than Alloy for all properties, thanks to the ability to control the solving process in ProB



(a) Varying the number of contexts (30 vertices, 12 contexts)

(b) Varying the number of vertices (13 rules, 10 contexts)

(c) Varying the number of rules (100 vertices, 30 contexts)

Fig. 3: Performance test results

by using B operations which allows one to determine an optimal order in which the constraints are solved, and also by storing frequently needed results into state variables of a B machine. Performance results are promising enough to consider ProB for the verification of real SGAC patient policies. The verification process is completely automatic.

To the extent of our knowledge, this is the first experiment that uses ProB on large data sets, uses rules in constraint solving, and uses B operations to guide the solving process. ProB has been previously used for verifying large data sets of railway parameters, but for some simpler formulas [15], and for university time tables [8]. Treating rules as objects for a constraint solver is a quite challenging task, as illustrated by the heavy computation times of Alloy.

Acknowledgements

We would like to thank M. Leuschel for his help and reactive support provided for ProB. This research was funded in part by CIUSSS-Estrie and by NSERC (Natural Sciences and Engineering Research Council of Canada). In particular, the authors would like to thank Hassan Diab, of CIUSSS-Estrie, and Mohammed Ouenzar, of Université de Sherbrooke, for their contribution in defining SGAC, and all the SGAC development team at UdeS and CIUSSS-Estrie.

References

[1] Jean-Raymond Abrial. *The B-book - assigning programs to meanings*. Cambridge University Press, 2005.

[2] Jeremy Bryans. Reasoning about XACML policies using CSP. In *SWS '05: Proceedings of the 2005 workshop on Secure web services*, pages 28–35. ACM Press, 2005.

[3] Frédéric Cuppens, Nora Cuppens-Boulahia, and Céline Coma. MotOrBAC: un outil d'administration et de simulation de politiques de sécurité. In *Security in Network Architectures (SAR) and Security of Information Systems (SSI), First Joint Conference*, pages 6–9, 2006.

[4] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *14th Int. Conf. TACAS 2008*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340, 2008.

[5] Morgan Deters, Andrew Reynolds, Tim King, Clark W. Barrett, and Cesare Tinelli. A tour of CVC4: how it works, and how to use it. In *IEEE Int. Conf. FMCAD 2014*, page 7, 2014.

[6] Anas Abou El Kalam, Salem Benferhat, Alexandre Miège, Rania El Baida, Frédéric Cuppens, Claire Saurel, Philippe Balbiani, Yves Deswarte, and Gilles Trouessin. Organization based access control. In *IEEE 4th Int. Workshop POLICY 2003*, pages 120–131, 2003.

[7] Jerome Falampin, Hung Le-Dang, Michael Leuschel, Mikael Mokrani, and Daniel Plagge. Improving railway data validation with ProB. In *Industrial Deployment of System Engineering Methods*, pages 27–43. Springer, 2013.

[8] Dominik Hansen, David Schneider, and Michael Leuschel. Using B and ProB for Data Validation Projects. In *5th Int. Conf. ABZ 2016*, volume 9675 of *Lecture Notes in Computer Science*, pages 167–182. Springer, 2016.

[9] Graham Hughes and Tevfik Bultan. Automated verification of access control policies using a SAT solver. *STTT*, 10(6):503–520, 2008.

[10] Nghi Huynh, Marc Frappier, Amel Mammar, and Régine Laleau. http://info.usherbrooke.ca/mfrappier/sgac/.

[11] Nghi Huynh, Marc Frappier, Amel Mammar, Régine Laleau, and Jules Desharnais. Validating the RBAC ANSI 2012 standard using B. In *5th Int. Conf. ABZ 2014*, Lecture Notes in Computer Science, pages 255–270, 2014.

[12] Nghi Huynh, Marc Frappier, Herman Pooda, Amel Mammar, and Regine Laleau. SGAC: A patient-centered access control method. In *10th IEEE Int. Conf. RCIS 2016*, pages 1–12, 2016.

[13] Daniel Jackson. *Software Abstractions: Logic, Language and Analysis*. MIT Press, 2012.

[14] Michael Leuschel and Michael J. Butler. ProB: an automated analysis toolset for the B method. *STTT*, 10(2):185–203, 2008.

[15] Michael Leuschel, Jérôme Falampin, Fabian Fritz, and Daniel Plagge. Automated property verification for large scale B models with ProB. *Formal Aspects of Computing*, 23(6):683–709, 2011.

[16] Mahdi Mankai and Luigi Logrippo. Access control policies: Modeling and validation. In *5th NOTERE Conference (Nouvelles Technologies de la Répartition)*, pages 85–91, 2005.

[17] Herman Pooda. Évaluation et comparaison des modèles de contrôle d'accès. Master's thesis, Université de Sherbrooke, 2015.

[18] David Power, Mark Slaymaker, and Andrew Simpson. On Formalizing and Normalizing Role-Based Access Control Systems. *The Computer Journal*, 52(3):305–325, 2009.

[19] Nafees Qamar, Yves Ledru, and Akram Idani. Validation of security-design models using Z. In *Proceedings of the 13th international conference on Formal methods and software engineering*, ICFEM'11, pages 259–274, Berlin, Heidelberg, 2011. Springer-Verlag.

[20] Erik Rissanen. *eXtensible Access Control Markup Language (XACML) Version 3.0*. OASIS, 2010.

[21] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control model. *IEEE Computer*, 29(2):38–47, 1996.