

# Validating SGAC Access Control Policies with Alloy and ProB

Nghi Huynh, Marc Frappier, Amel Mammar and Régine Laleau

FA 2018, April 30th



UNIVERSITÉ DE  
**SHERBROOKE**

- 1 Introduction
- 2 SGAC
- 3 Formalization
- 4 Automated verification
- 5 Conclusion

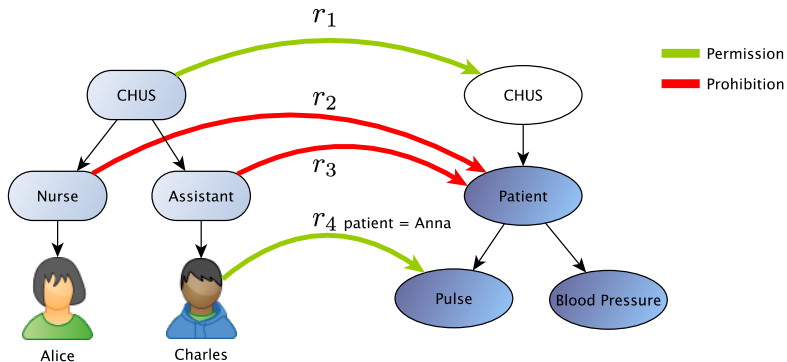
## Motivation

- Consent Management in Electronic Health Records
- Hospital of Université de Sherbrooke (CHUS) in Québec, Canada.
- Two major stakes in access control (healthcare) :
  - 1) patient privacy → consent
  - 2) patient safety → ????????

# Presentation of SGAC

- SGAC = Automated Consent Management System
- Designed to meet CHUS requirements
- Features:
  - hierarchy among users;
  - hierarchy among data;
  - explicit prohibitions;
  - automated conflict resolutions.

# Example



## Conflict Resolution Strategy

$r_a$  has precedence over  $r_b$  iff:

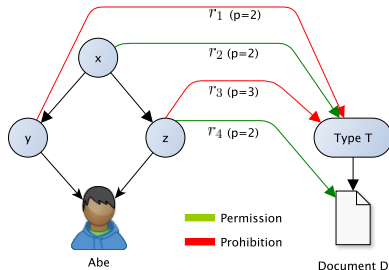
- 1  $r_a$ 's priority value is lower;  
 ex:  $r_2$  has precedence over  $r_3$ .

or

- 2 same priority and  
 $r_a$ 's subject is more specific;  
 ex:  $r_1$  has precedence over  $r_2$ .

or

- 3 same priority and  
 incomparable subjects,  
 and  $r_a.m = -r_b.m = +$ .  
 ex:  $r_1$  has precedence over  $r_4$ .



## Conflict Resolution Strategy

$r_a$  has precedence over  $r_b$  iff:

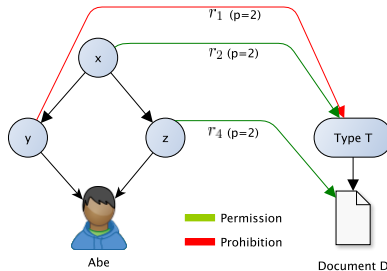
- 1  $r_a$ 's priority value is lower;  
 ex:  $r_2$  has precedence over  $r_3$ .

or

- 2 same priority and  
 $r_a$ 's subject is more specific;  
 ex:  $r_1$  has precedence over  $r_2$ .

or

- 3 same priority and  
 incomparable subjects,  
 and  $r_a.m = -r_b.m = +$ .  
 ex:  $r_1$  has precedence over  $r_4$ .



## Conflict Resolution Strategy

$r_a$  has precedence over  $r_b$  iff:

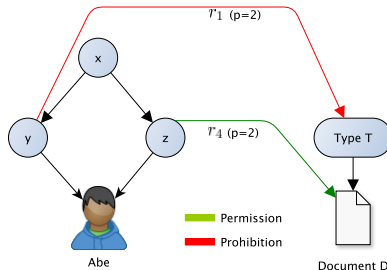
- 1  $r_a$ 's priority value is lower;  
 ex:  $r_2$  has precedence over  $r_3$ .

or

- 2 same priority and  
 $r_a$ 's subject is more specific;  
 ex:  $r_1$  has precedence over  $r_2$ .

or

- 3 same priority and  
 incomparable subjects,  
 and  $r_a.m = -r_b.m = +$ .  
 ex:  $r_1$  has precedence over  $r_4$ .





## Conflict Resolution Strategy

$r_a$  has precedence over  $r_b$  iff:

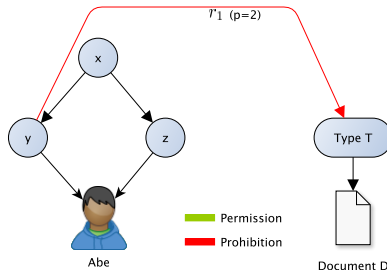
- 1  $r_a$ 's priority value is lower;  
 ex:  $r_2$  has precedence over  $r_3$ .

or

- 2 same priority and  
 $r_a$ 's subject is more specific;  
 ex:  $r_1$  has precedence over  $r_2$ .

or

- 3 same priority and  
 incomparable subjects,  
 and  $r_a.m = -r_b.m = +$ .  
 ex:  $r_1$  has precedence over  $r_4$ .



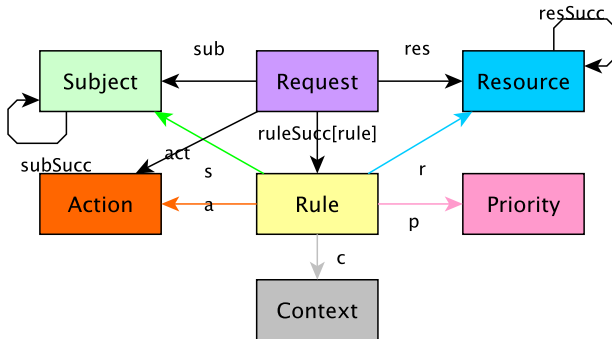
# Properties

The properties we want to check are:

- access:  
can health worker  $W$  have access to the document  $D$  ?
- ineffective rule detection:  
what are the rules that are never taken into account when evaluating a request ?
- important hidden data detection:  
are there important data that are unreachable by any health worker ?
- granting context detection:  
in which contexts is a given request granted ?

# Formalization

Huynh *et al.*, SGAC: A patient-centered access control method, (RCIS'16).



## Rule ordering

$r_a$  has precedence over  $r_b$  iff:

- 1  $r_a$ 's priority value is lower;  
ex:  $r_2$  has precedence over  $r_3$ .

or

- 2 same priority and  
 $r_a$ 's subject is more specific;  
ex:  $r_1$  has precedence over  $r_2$ .

or

- 3 same priority and  
incomparable subjects,  
and  $r_a.m = - r_b.m = +$ .  
ex:  $r_1$  has precedence over  $r_4$ .

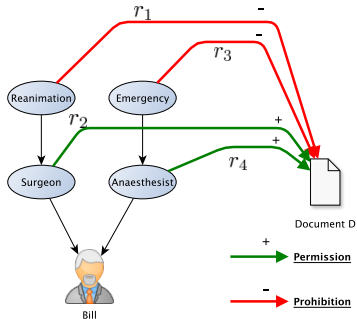
Two steps :

- introduction of ' $\prec$ ': ordering with priority and subject specificity (phase 1-2);
- introduction of ' $<$ ': final ordering (phase 3).

## Why two steps?

Only maximal elements of  $\prec$  must be compared with their modality.

ex: without the maximal element condition,  
 $r_1 < r_2$ ,  $r_3 < r_4$ ,  $r_2 < r_3$  and  $r_4 < r_1$ .



# Request Evaluation

In order to evaluate a request in a given context :

- 1 we select all applicable rules to the request;
- 2 we order the applicable rules;
- 3 we analyse the graph made of the ordered rules : the sinks of the graph determine the result of the request.

## Automated verification

We use first order logic based tools : Alloy and ProB.

### Alloy

Alloy is a model finder that offers a graphical interface and evaluator that are very useful to debug and help understandings counter-examples.

### ProB

ProB is a model checker and animator for the B method. Its constraint solving capability allows it to do model finding.

## Let's get started : simplifications first !

In order to be able to conduct tractable verification with the tools, we have to make some adjustments:

- reduce the size of the graphs: verification is done for each patient, thus resource graph can be cut ;
- ignore the actions: the approach taken for each action is the same;
- reduce computational burden: with the current approach, a graph is built for each context+request  
→ 1 request = 1 graph.



# Alloy

## Difficulty

Alloy cannot handle the number of requests ( $|PERSON \times DOCUMENTS|$ ).

## Solution

Explicitly define one request at a time. The others target also persons and documents but are left undetermined.

## Results

Alloy can conduct the verification, but some properties cannot be directly verified.

# ProB

## Difficulty

ProB does not manage to process and order the rules for all the requests.

## Solution

Program and guide the variable calculus order.  
Ex: process  $\prec$  et  $<$  successively and separately.

## Results

ProB finally manages to order the rules, and this solution provides a way to reduce further the processing time.

# ProB

## Difficulty

How can we encode efficiently the properties ?

## Solution

Properties are encoded into the operations of each machine. For instance,  $access(req, con)$

- precondition : arguments  $req$  and  $con$  are a request and a context.
- postcondition : result of  $req$  within the context  $con$ .

## Results

- Verification is done for all possible combinations;
- All properties are verified in only one run.

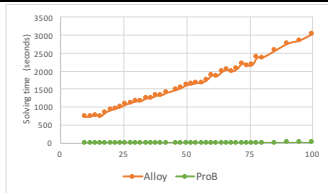
# Performance Test

## Process

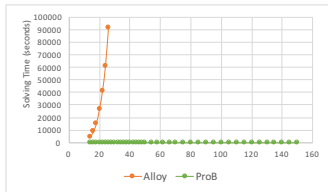
Set all parameters except one which we vary.

For each configuration (number of vertices, of rules, of contexts) :

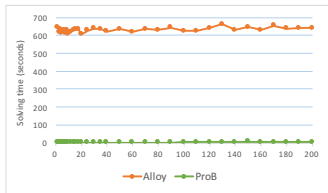
- ① Generation of random graphs and random rules;
- ② For each graph, random requests are picked;
- ③ For each request :
  - access property in random contexts;
  - detection of granting contexts;
  - detection hidden documents;
  - detection ineffective rules.



*13 rules, 10 contexts :*  
linear processing time



*100 vertices, 30 contexts :*  
exponential processing time



*30 vertices, 12 rules :*  
quasi-constant processing time

## Conclusion

- ProB outperforms Alloy thanks to the ability to 'program' how the computations are done
- Automated verification in real cases can be conducted (offline) with ProB
  - 300 vertices, 160 rules, 100 contexts with 200 requests in about 15 minutes with ProB
- Alloy is better than ProB in brute force in several cases, but it is insufficient here
- Need similar ability in Alloy to program the model finding
  - Integrate  $\alpha$ Ruby in Alloy

## Questions

Thanks for your attention !