Removing Problems in Rule-Based Policies

Zheng Cheng Jean-Claude Royer Massimo Tisi

IMT Atlantique, NaoMod Team, LS2N
firstname.lastname@imt-atlantique.fr

June, 27 2019

2019-06-19

Removing Problems in Rule-Based Policies

Removing Problems in Rule-Based Policies

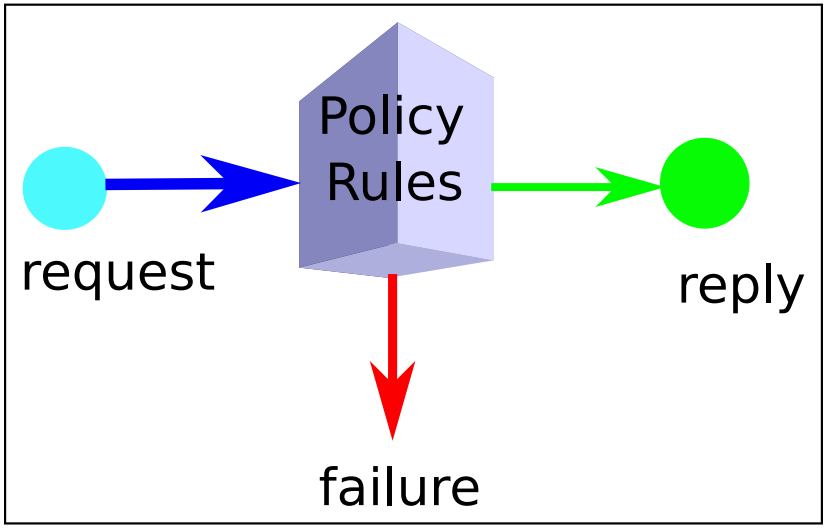
Zheng Cheng Jean-Claude Royer Massimo Tisi

IMT Atlantique, NaoMod Team, LS2N
firstname.lastname@imt-atlantique.fr

June, 27 2019

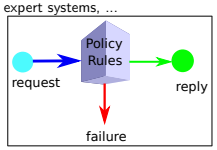
Context

1. General rule based systems with a logical perspective
2. This formalism is used in many contexts: logic programming, data bases, expert systems, ...



3. Our focus is **Security Policy**
4. Finding all conflicts or **undefined requests** as in [1]
5. Resolving these problems
6. FOL with a decision procedure for satisfiability

Context
- General rule based systems with a logical perspective
- This formalism is used in many contexts: logic programming, data bases, expert systems, ...



- Our focus is **Security Policy**
- Finding all conflicts or **undefined requests** as in [1]
- Resolving these problems
- FOL with a decision procedure for satisfiability

Our context is rule-based systems with a logical perspective.
These systems are popular and used in many applications: expert systems, logic programming, data bases, and security policies to name a few.
Abstractly we can consider a rule system as an engine which receives requests and gives a reply or a failure.
Our objective is to extend our previous work in which we find all the conflicts in such a system. Now we expect to automatically resolve some of these problems.

We assume FOL we a decision procedure for satisfiability.

A Motivating Example

Listing 1: Rules for some hospital roles

```
1  And(Patient(T, X), PrimaryDoctor(T, X)) => False
2  And(Receptionist(T, X), Doctor(T, X)) => False
3  And(Nurse(T, X), Doctor(T, X)) => False
4  Nurse(T, X) => Employee(T, X)
5  Doctor(T, X) => Employee(T, X)
6  Receptionist(T, X) => Employee(T, X)
7  MedicalManager(T, X) => Employee(T, X)
8  Manager(T, X) => Employee(T, X)
9  Patient(T, X) => PatientWithTPC(T, X)
10 Doctor(T, X) => ReferredDoctor(T, X)
11 Doctor(T, X) => PrimaryDoctor(T, X)
```

- `Exists([T, X], And(Doctor(T, X), Not(Nurse(T, X))))` SAT
- `Exists([T, X], And(Doctor(T, X), Not(Nurse(T, X))), Patient(T, X)))` UNSAT

A Motivating Example

Listing 1: Rules for some hospital roles

```
1  And(Patient(T, X), PrimaryDoctor(T, X)) => False
2  And(Receptionist(T, X), Doctor(T, X)) => False
3  And(Nurse(T, X), Doctor(T, X)) => False
4  Nurse(T, X) => Employee(T, X)
5  Doctor(T, X) => Employee(T, X)
6  Receptionist(T, X) => Employee(T, X)
7  MedicalManager(T, X) => Employee(T, X)
8  Manager(T, X) => Employee(T, X)
9  Patient(T, X) => PatientWithTPC(T, X)
10 Doctor(T, X) => ReferredDoctor(T, X)
11 Doctor(T, X) => PrimaryDoctor(T, X)
```

- `Exists([T, X], And(Doctor(T, X), Not(Nurse(T, X))))` SAT
- `Exists([T, X], And(Doctor(T, X), Not(Nurse(T, X))), Patient(T, X)))` UNSAT

We have here an example extract from an healthcare policy.
Rule are logical implications written in a style close to the Z3 solver.
These rules describe hierarchical and exclusion relations between few roles in an hospital.
A conflict is a request leading to contradictory replies.
Our previous work published at IFM try to find all the conflicts.

All the Undefined Requests

Listing 2: The roles module analysis

```
 ---------- UNSAFE -------------
[0, 0, 0, 1, 1, 1, -1, -1, -1, -1, -1]
  And(Not(Nurse(T, X)), Doctor(T, X),
      Not(PrimaryDoctor(T, X)),
      Not(Receptionist(T, X)), Patient(T, X)) => False
[1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
  And(Patient(T, X), PrimaryDoctor(T, X)) => False

... another 2 unsafe exclusive rules
 ---------- NOT UNSAFE -------------
[0, 0, 0, 1, 0, 0, 1, -1, -1, -1, -1]
  And(Nurse(T, X), Not(Doctor(T, X)), Patient(T, X), Not(PrimaryDoctor(T
     , X)))
     => And(PatientWithTPC(T, X), Employee(T, X))

... another 9 not unsafe exclusive rules
```

All the Undefined Requests

Listing 2: The roles module analysis

```
 ---------- UNSAFE --------------
[0, 0, 0, 1, 1, 1, -1, -1, -1, -1, -1]
  And(Not(Nurse(T, X)), Doctor(T, X),
      Not(PrimaryDoctor(T, X)),
      Not(Receptionist(T, X)), Patient(T, X)) => False
[1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
  And(Patient(T, X), PrimaryDoctor(T, X)) => False

... another 2 unsafe exclusive rules
 ---------- NOT UNSAFE --------------
[0, 0, 0, 1, 0, 0, 1, -1, -1, -1, -1]
  And(Nurse(T, X), Not(Doctor(T, X)), Patient(T, X), Not(PrimaryDoctor(T
     , X)))
     => And(PatientWithTPC(T, X), Employee(T, X))

... another 9 not unsafe exclusive rules
```

The output with the previous example will be a set of exclusive rules which are equivalent to the original system.
There are two parts the unsafe and the not unsafe.
The main difference is that we are sure that unsafe rules are leading to what we call undefined requests or problems here.
That is requests which in conjunction with the original system makes it unsatisfiable.
Each of these rule is characterized by a binary number which represent the combinations of the original rules leading to this rule.
For instance the second example comes from only one original rule which was unsafe in the previous example and describe some constraints given by the specifier.
The first rule describes a more complex problem not made explicit in the original example and surely a bug the specifier should resolve.
We will focus on removing problems from the unsafe rules but not unsafe rules are also describing undefined requests and our work also apply to

Objective

1. **Removing** a selection of these undefined requests
2. This means that after removing, the request will be defined
3. This is simpler than **fixing** which needs human interaction to give the good response
4. While removing is completely automatic
5. This is a mandatory step before fixing
6. Note that this is not sensible to remove all (since we will get a useless tautological system)
7. Choice of the undefined requests to remove: a subset of **those in unsafe rules**
8. But it applies to other cases

Objective

1. **Removing** a selection of these undefined requests
2. This means that after removing, the request will be defined
3. This is simpler than **fixing** which needs human interaction to give the good response
4. While removing is completely automatic
5. This is a mandatory step before fixing
6. Note that this is not sensible to remove all (since we will get a useless tautological system)
7. Choice of the undefined requests to remove: a subset of **those in unsafe rules**
8. But it applies to other cases

Resolving an undefined request or a problem means here removing it.
It means that after removing the request will be defined thus in conjunction with the new system it will be satisfiable.
This is simpler than fixing a problem which needs in general human interaction to provide the good reply associated to the input request.
But removing is automatic and can be viewed as a mandatory step before to tackle the fixing task.
Note also that it is not sensible to remove all the undefined requests else we get a useless tautological system.
Thus a choice of the problems to solve is required, here we will focus on those made explicit by the unsafe rules.

Related Work

- Automatic software/system repair
- [2] for a review
- Redundancies [3], misconfigurations [4]
- Conflicts and inconsistencies in rule-based systems
- [5] will govern rule applications to have compatible replies
- Defining conflict-free meta-rules [3].
- Son et al. [6] repair access-control policies in web applications
- Wu focuses on detecting inconsistency bugs among invariant rules enforced on UML models [7]
- Our approach is original: consider all the conflicts, minimal modification of rules, time optimization

Related Work

- Automatic software/system repair
- [2] for a review
- Redundancies [3], misconfigurations [4]
- Conflicts and inconsistencies in rule-based systems
- [5] will govern rule applications to have compatible replies
- Defining conflict-free meta-rules [3].
- Son et al. [6] repair access-control policies in web applications
- Wu focuses on detecting inconsistency bugs among invariant rules enforced on UML models [7]
- Our approach is original: consider all the conflicts, minimal modification of rules, time optimization

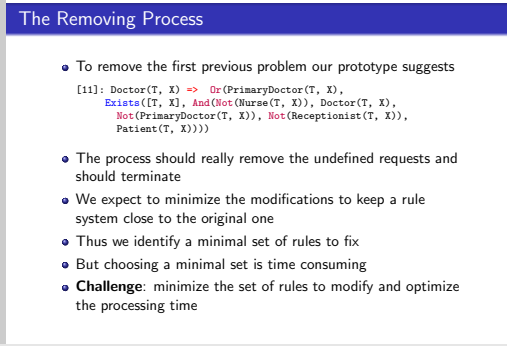
Our work is related to automatic code repair and a good review has been done by M.Montperrus.
There are several work around redundancies or misconfigurations, but we focus on conflicts or inconsistencies.
TODO review cuppens + Hu
Son
There are also work related to fixing UML class diagram
We think that our work is original since we first consider all conflicts allowing a choice of the most critical, and we try to localize the modifications and to minimize the processing time.

The Removing Process

- To remove the first previous problem our prototype suggests

```
[11]: Doctor(T, X) =>  Or(PrimaryDoctor(T, X),
      Exists([T, X], And(Not(Nurse(T, X)), Doctor(T, X),
        Not(PrimaryDoctor(T, X)), Not(Receptionist(T, X)),
        Patient(T, X))))
```

- The process should really remove the undefined requests and should terminate
- We expect to minimize the modifications to keep a rule system close to the original one
- Thus we identify a minimal set of rules to fix
- But choosing a minimal set is time consuming
- **Challenge**: minimize the set of rules to modify and optimize the processing time



Our prototype given the previous example will suggest to change the 11th rule.

The new rule is the old one with in the conclusion an additional expression which is the problem to remove.

The removing process should effectively remove the chosen problems and should terminate.

Termination is not a real problem but we expect to minimise the modifications of the rule system.

In fact we change some rules thus we need to identify a minimal set which remove the problem.

But to do that we do not find another way than testing the possible choices.

Thus our challenge is to minimize the set of rules and to try to decrease the processing time.

Principle and Properties

- We do not discuss here simpler alternative like adding new explicit unsafe rules and other tricks
- R is $\forall * \bigwedge_{1 \leq i \leq n}(D_i => C_i)$ and If U is an undefined request to exclude the fix is: $\neg U => R$
- Which is equivalent to $\forall * \bigwedge_{1 \leq i \leq n}(D_i => (C_i \vee U))$
- Let F the selected set of rules to modify, $R/F/U$ the modified system and $R_{\neg F}$ the non modified part of R
- $R/F/U = R \vee (U \wedge R_{\neg F})$
- **Effectiveness:** removing the problem U does not add new problems, since we have $\neg(R/F/U) => \neg R$
- **Behaviour preservation:** if $req => R$ the behaviour is preserved for any problem and any selection F

Principle and Properties

- We do not discuss here simpler alternative like adding new explicit unsafe rules and other tricks
- R is $\forall * \bigwedge_{1 \leq i \leq n}(D_i => C_i)$ and If U is an undefined request to exclude the fix is: $\neg U => R$
- Which is equivalent to $\forall * \bigwedge_{1 \leq i \leq n}(D_i => (C_i \vee U))$
- Let F the selected set of rules to modify, $R/F/U$ the modified system and $R_{\neg F}$ the non modified part of R
- $R/F/U = R \vee (U \wedge R_{\neg F})$
- **Effectiveness:** removing the problem U does not add new problems, since we have $\neg(R/F/U) => \neg R$
- **Behaviour preservation:** if $req => R$ the behaviour is preserved for any problem and any selection F

There are some simple way to remove some problems but they are not general and have drawbacks.
Thus here we use a general principle expressed by this formula: the negation of the problem to remove becomes a guard of the rule system.
It is easy to see that this is equivalent to add the problem in the conclusion of all the rules.
We note F the selection of rules to modify, $R/F/U$ the new system resulting from the addition of U in all the F rules and R_notF the non modified rules of R.
From this equivalence we can analyze two desirable properties.
The first say that removing a problem U does not add new problems.
This is quite easy to see because if U' is a problem after the modification it is a problem already in U.
The second property is: What behaviours of the original rule system could rule developers expect to preserve after the removing process applied. But behaviours preservation is true for request which are included in R which

Finding the Minimal Selection F

- Modifying principle: To add $\neg U$ in the selected rule conditions or **to add U in the rule conclusions**
- Latter is simpler since we have the same enumerative decomposition for R and $R/F/U$
- **Correct Fix of a Rule System**: $R/F/U$ is a correct fix for R with F and U if each rule in F has its conclusion enlarged with U and U is not a problem for $R/F/U$.
- Note that: If F is a correct fix then $F \subset G$ is also a correct fix
- **Naive Check**: $R/F/U$ is a correct fix if and only if $U \wedge R_{\neg F}$ is satisfiable

Finding the Minimal Selection F

- Modifying principle: To add $\neg U$ in the selected rule conditions or **to add U in the rule conclusions**
- Latter is simpler since we have the same enumerative decomposition for R and $R/F/U$
- **Correct Fix of a Rule System**: $R/F/U$ is a correct fix for R with F and U if each rule in F has its conclusion enlarged with U and U is not a problem for $R/F/U$.
- Note that: If F is a correct fix then $F \subset G$ is also a correct fix
- **Naive Check**: $R/F/U$ is a correct fix if and only if $U \wedge R_{\neg F}$ is satisfiable

Indeed removing a problem in a rule can also be done by adding the negation of U in the rule condition.
Latter is simpler since we have the same enumerative decomposition for R and $R/F/U$ and we can reuse our previous work at IFM.
The definition of a correct fix is rather straightforward from that.
We note that If F is a correct fix then any G super set of F is also a correct fix.
Of course the set of all the rules is a correct fix.
Then we have a first algorithm, called naive check, which checks if the problem to remove is satisfiable in conjunction with the non modified part of the system.

Single or complex problem

- If U is a problem for R with a set of conditions $D_{1 \leq i \leq n}$ then $U => \bigvee_{j \in J} \exists * D_j$, where J is a subset of $1 \leq i \leq n$
- Rules built by the enumerative methods are exclusive since conditions are $(\bigwedge_{i \in I_1} D_i \bigwedge_{j \in I_0} \neg D_j)$
- I_1 for positive rules and I_0 for negative ones
- Thus U can be split in disjoint parts each one associated to a binary characteristic
- **Single (complex) problem**: the binary characteristic does not have don't care bit: [1 0 1 0 1] (eg. [1 0 -1 -1 1])
- Let U a satisfiable problem such that
  $U => \exists * \ (\bigwedge_{i \in I_1} D_i \bigwedge_{j \in I_0} \neg D_j)$ then
  $U = U \wedge (\forall * \ (\bigwedge_{i \in I_1} D_i \bigwedge_{j \in I_0} \neg D_j)$ **(Universal part)**
  $\vee \ U \wedge (\exists * \ (\bigvee_{i \in I_1} \neg D_i \bigvee_{j \in I_0} D_j)$ **(Existential part)**

Single or complex problem

- If U is a problem for R with a set of conditions $D_{1 \leq i \leq n}$ then $U => \bigvee_{j \in J} \exists * D_j$, where J is a subset of $1 \leq i \leq n$
- Rules built by the enumerative methods are exclusive since conditions are $(\bigwedge_{i \in I_1} D_i \bigwedge_{j \in I_0} \neg D_j)$
- I_1 for positive rules and I_0 for negative ones
- Thus U can be split in disjoint parts each one associated to a binary characteristic
- **Single (complex) problem**: the binary characteristic does not have don't care bit: [1 0 1 0 1] (eg. [1 0 -1 -1 1])
- Let U a satisfiable problem such that $U => \exists * \ (\bigwedge_{i \in I_1} D_i \bigwedge_{j \in I_0} \neg D_j)$ then $U = U \wedge (\forall * \ (\bigwedge_{i \in I_1} D_i \bigwedge_{j \in I_0} \neg D_j)$ **(Universal part)** $\vee \ U \wedge (\exists * \ (\bigvee_{i \in I_1} \neg D_i \bigvee_{j \in I_0} D_j)$ **(Existential part)**

Since we expect to improve it we find another solution.
A problem is included in the negation of the rule system thus it means it is included in the union of some rule conditions.
But if we have exclusive rule we can characterize a problem by the positive and negative rule condition. Thus we can define a notion of single or complex problem depending if the union of this two sets overlaps the set of rules.
Indeed the complex case if the general case for a problem.
Such a complex problem can be split in two parts called respectively universal and existential as shown by this formula.
The benefit here is that if we consider the universal part it only triggers some rules thus we get a sufficient condition for satisfiability.

Removing criterion

- Sufficient criterion based on checking the universal part only
- A single universal problem triggers only one rule in the enumeration of $R/F/U$
- **Removing Criterion for Single Problem**: Let U a single problem with positive rules I_1 thus $R/F/U$ is a correct fix if either $I_1 \subset F$ or
  $I_1 \cap F \neq \emptyset$ and $U \wedge \forall * (\bigwedge_{i \in I_1} D_i \bigwedge_{j \in I_0} \neg D_j \bigwedge_{i \in I_1 \cap \neg F} C_i)$ is satisfiable
- **Removing Criterion for a Complex Problem**: Let U a complex problem, $R/F/U$ is a correct fix if $\neg F \subset I_0$ or
  $F \cap \neg I_0$ and
  $U \wedge \forall * (\bigwedge_{i \in I_1} D_i \bigwedge_{j \in I_0} \neg D_j \bigwedge_{i \in I_1 \cap \neg F} C_i) \bigwedge_{g \in \neg I_1 \cap I_0 \cap \neg F} ((\forall * D_g \forall * C_g) \vee \forall * \neg D_g)$ is satisfiable
- Equivalence condition needs satisfiability of the union of the universal and existential parts

Removing criterion

- Sufficient criterion based on checking the universal part only
- A single universal problem triggers only one rule in the enumeration of $R/F/U$
- **Removing Criterion for Single Problem**: Let U a single problem with positive rules I_1 thus $R/F/U$ is a correct fix if either $I_1 \subset F$ or
  $I_1 \cap F \neq \emptyset$ and $U \wedge \forall * (\bigwedge_{i \in I_1} D_i \bigwedge_{j \in I_0} \neg D_j \bigwedge_{i \in I_1 \cap \neg F} C_i)$ is satisfiable
- **Removing Criterion for a Complex Problem**: Let U a complex problem, $R/F/U$ is a correct fix if $\neg F \subset I_0$ or
  $F \cap \neg I_0$ and
  $U \wedge \forall * (\bigwedge_{i \in I_1} D_i \bigwedge_{j \in I_0} \neg D_j \bigwedge_{i \in I_1 \cap \neg F} C_i) \bigwedge_{g \in \neg I_1 \cap I_0 \cap \neg F} ((\forall * D_g \forall * C_g) \vee \forall * \neg D_g)$ is satisfiable
- Equivalence condition needs satisfiability of the union of the universal and existential parts

If we consider only the universal part of the problem we get a sufficient condition for the definedness after modifying the rule system.
For a single problem it is based on the fact that the universal part will trigger only one rule in the enumeration of R/F/U/.
We have a similar criterion for a complex problem which is obtained by a completion of the binary characteristic and properties of FOL.
These are the criterion we use in our algorithms to search for a minimal set of rules sufficient to remove a problem.
And if we expect a general equivalence we should also consider the satisfiability of the existential part in conjunction with the non modified rules.

Looking for Minimal Size

- Two processes: top-down or bottom-up, worst complexity is exponential
- Bottom-up is better since it stops once the solution is found
- Minimal core satisfiability does not respect the rule structure
- Naive approach is straightforward
- `lookup_complex` algorithm checking first the universal part of U
- We search F inside all the positive rules in the binary and its completion, thus look for less combinations than the naive approach
- Checking the universal part first, and relying on the transition phase phenomenon [8]
- Satisfiability checking of the universal formula
- CNF has a ratio in the probably satisfiable area in a small amount of time

Looking for Minimal Size

- Two processes: top-down or bottom-up, worst complexity is exponential
- Bottom-up is better since it stops once the solution is found
- Minimal core satisfiability does not respect the rule structure
- Naive approach is straightforward
- `lookup_complex` algorithm checking first the universal part of U
- We search F inside all the positive rules in the binary and its completion, thus look for less combinations than the naive approach
- Checking the universal part first, and relying on the transition phase phenomenon [8]
- Satisfiability checking of the universal formula
- CNF has a ratio in the probably satisfiable area in a small amount of time

Indeed two search processing are possible: top-down or bottom-up.
We expect that the minimal is close to the bottom.
While the top-down m is found, must prove the minimality of it by checking all the smaller combinations in the next level.
The implementation of the naive approach is straightforward.
Our lookup algorithm is more efficient for two reasons.
The first is that we look for less combinations since we look for subset of positive rules in the binary characteristic.
The second reason is due to the universal formula and it is surely less general we check it first since it seems often sufficient and more efficient.
In satisfiability there is a transition phase phenomenon which says that below a certain threshold satisfiability is efficiently proved true.
This threshold is a ratio of the maximal number of clauses and the number of literal in the clauses.

We estimate that with the universal formula we are below this threshold.

Application to the Healthcare Example

- Translation of the original example from [9]
- An administrative RBAC policy with role assignments and revocations
- `http://www3.cs.stonybrook.edu/~stoller/ccs2007/`
- FOL with discrete time, integer comparison, arity in [1 .. 3], 61 rules
- Can be structured in four modules: roles, permissions, assignment and revocations
- First: simple specification of the assignment module (effect at $T + 1$)
- In isolation: no new unsafe rules in isolation and together
- Results: safe= 20817 unsafe= 3 time: 6327s

Application to the Healthcare Example

- Translation of the original example from [9]
- An administrative RBAC policy with role assignments and revocations
- `http://www3.cs.stonybrook.edu/~stoller/ccs2007/`
- FOL with discrete time, integer comparison, arity in [1 .. 3], 61 rules
- Can be structured in four modules: roles, permissions, assignment and revocations
- First: simple specification of the assignment module (effect at $T+1$)
- In isolation: no new unsafe rules in isolation and together
- Results: safe= 20817 unsafe= 3 time: 6327s

We think that because the original example was rather well designed we do not found too much unsafe rules.

Second Specification

- An alternative specification of the `assignment` module is to write rules without changing the time instant in the conclusion
- It generates "unexpected" conflicts: we get 91 not unsafe rules and three unsafe rules in nearly 8s
- Thus using our `lookup_complex` procedure we find that these problems are all removed by modifying the rule: the 5^{th} rule
- The enumerative computation of the new module shows that it has no more unsafe rule
- All together: the roles module has new unsafe rules with the assignment module, indeed there are 3 new unsafe rules
- Minimal fix with the 4^{th} rule
- The result was computed in nearly 5200s and generates 20817 not unsafe rules and the three explicit unsafe rules from the roles module

Second Specification

- An alternative specification of the `assignment` module is to write rules without changing the time instant in the conclusion
- It generates "unexpected" conflicts: we get 91 not unsafe rules and three unsafe rules in nearly 8s
- Thus using our `lookup_complex` procedure we find that these problems are all removed by modifying the rule: the 5^{th} rule
- The enumerative computation of the new module shows that it has no more unsafe rule
- All together: the roles module has new unsafe rules with the assignment module, indeed there are 3 new unsafe rules
- Minimal fix with the 4^{th} rule
- The result was computed in nearly 5200s and generates 20817 not unsafe rules and the three explicit unsafe rules from the roles module

The ContinueA Example

- Translation of the example originally used in [10]
- `http://cs.brown.edu/research/plt/software/margrave/versions/01-01/examples/`
- XACML access control policy for a conference management system
- 47 FOL rules, arity in [1 .. 2]
- Do not **translate combining algorithms**
- A great amount of problems amongst them 530 unsafe rules while the number of not unsafe rules is 302 (computed in 97 seconds)
- We observed that the minimal set of fixing rules is generally low (between 1 and 5 rules)
- Costly to go up to more 4 rules
- Same minimal size than with the naive approach but few differences due to ordering search

The ContinueA Example

- Translation of the example originally used in [10]
- `http://cs.brown.edu/research/plt/software/margrave/versions/01-01/examples/`
- XACML access control policy for a conference management system
- 47 FOL rules, arity in [1 .. 2]
- Do not **translate combining algorithms**
- A great amount of problems amongst them 530 unsafe rules while the number of not unsafe rules is 302 (computed in 97 seconds)
- We observed that the minimal set of fixing rules is generally low (between 1 and 5 rules)
- Costly to go up to more 4 rules
- Same minimal size than with the naive approach but few differences due to ordering search

Measures for the two Policies

Usecase	Naive		Lookup		Additional measures		
	NS	NT	LS	LT	PR	DS	TF
Healthcare	1	1.01s	1	0.2s	10.1	0	509%
ContinueA	1.53	115s	1.53	31s	3.9	0	794%

- Naive approach: the mean of minimal size (NS), mean of time (NT)
- Lookup: the same for the lookup method with LS, LT
- PR: the mean of positive rules in each problem
- DS: the maximum of differences between size of the selection
- TF: the mean of the ratio: naive time divided by lookup time

Measures for the two Policies

Usecase	Naive		Lookup		Additional measures		
	NS	NT	LS	LT	PR	DS	TF
Healthcare	1	1.01s	1	0.2s	10.1	0	509%
ContinueA	1.53	115s	1.53	31s	3.9	0	794%

- Naive approach: the mean of minimal size (NS), mean of time (NT)
- Lookup: the same for the lookup method with LS, LT
- PR: the mean of positive rules in each problem
- DS: the maximum of differences between size of the selection
- TF: the mean of the ratio: naive time divided by lookup time

We get the same minimal size but sometimes we do not have the same selection to fix due to the ordering of the searching process.

Selection Distribution and Mean Time

F size	Frequency	Naive	Lookup	Time factor
1	63%	0.33s	0.04s	800%
2	27%	6.6s	1.15s	573%
3	8%	124s	24s	517%
4	3%	1573s	281s	560%
5	0.5 %	88890	3433	256%

F size	Frequency	Naive	Lookup	Time factor
1	63%	0.33s	0.04s	800%
2	27%	6.6s	1.15s	573%
3	8%	124s	24s	517%
4	3%	1573s	281s	560%
5	0.5 %	88890	3433	256%

This distribution shows that most of the problems are fixed modifying 1 or two rules.
But it is costly to go up to 4 rules. And processing top-down is not really possible in most of the examples.

Discussion

- In the healthcare examples we remove only few undefined requests coming from the unsafe rules
- For the ContinueA example removing all the unsafe problems in 47 rules
  - Naive approach: $17 hours$ and the increase in size of $24910 * US$ where US is the median size of the problems
  - Lookup algorithm: $4.6 hours$ and the increase in size is $796 * US$.
- Undefined requests in not unsafe rules can be also removed
  - for the 320 cases in ContinueA we get a mean for $TF = 120\%$
  - with 3000 problems (nearly 10% of the problems) of the Healthcare we get a mean of 800%.
- The existential part has a greater impact here

Discussion

- In the healthcare examples we remove only few undefined requests coming from the unsafe rules
- For the ContinueA example removing all the unsafe problems in 47 rules
  - Naive approach: 17*hours* and the increase in size of $24910 * US$ where US is the median size of the problems
  - Lookup algorithm: 4.6*hours* and the increase in size is $796 * US$.
- Undefined requests in not unsafe rules can be also removed
  - for the 320 cases in ContinueA we get a mean for $TF = 120\%$
  - with 3000 problems (nearly 10% of the problems) of the Healthcare we get a mean of 800%.
- The existential part has a greater impact here

Conclusion

- We study how to make undefined request defined
- We analyze the problem in general
- Challenge: minimize the set of rules to modify while optimizing the processing time
- It is successful on two middle-size use cases
- Seems to work on complex FOL example not only pure freely quantified predicate calculus
- More experiments are needed to enlarge applicability
- Future: computing a simplified summary of all the real problems

Conclusion

- We study how to make undefined request defined
- We analyze the problem in general
- Challenge: minimize the set of rules to modify while optimizing the processing time
- It is successful on two middle-size use cases
- Seems to work on complex FOL example not only pure freely quantified predicate calculus
- More experiments are needed to enlarge applicability
- Future: computing a simplified summary of all the real problems

└Conclusion

In fact we did another variant of the healthcare specification.
The reason is that we saw also +++

QUESTIONS ?

QUESTIONS ?


Zheng Cheng, Jean-Claude Royer, and Massimo Tisi.
Efficiently characterizing the undefined requests of a rule-based system.
In *IFM 2018*, 2018.


Martin Monperrus.
Automatic software repair: A bibliography.
ACM Computing Surveys, 51(1):17:1–17:24, 2018.


Hongxin Hu, Gail-Joon Ahn, and Ketan Kulkarni.
Discovery and resolution of anomalies in web access control policies.
IEEE Transactions on Dependable and Secure Computing, 10(6):341–354, 2013.


Joaquin Garcia-Alfaro, Frédéric Cuppens, Nora Cuppens-Boulahia, Salvador Martinez, and Jordi Cabot.
Management of stateful firewall misconfiguration.

Computers and Security, 39:64–85, 2013.


Frédéric Cuppens, Nora Cuppens-Boulahia, Joaquín García-Alfaro, Tarik Moataz, and Xavier Rimasson.
Handling stateful firewall anomalies.
In Dimitris Gritzalis, Steven Furnell, and Marianthi Theoharidou, editors, *Information Security and Privacy Research*, volume 376 of *IFIP Advances in Information and Communication Technology*, pages 174–186. Springer, 2012.


Sooel Son, Kathryn S. McKinley, and Vitaly Shmatikov.
Fix me up: Repairing access-control bugs in web applications.
In *20th Annual Network and Distributed System Security Symposium*, San Diego, California, USA, 2013. Usenix.


Hao Wu.
Finding achievable features and constraint conflicts for inconsistent metamodels.

In *13th European Conference on Modelling Foundations and Applications*, pages 179–196, Marburg, Germany, 2017. Springer.

Dimitris Achlioptas, Assaf Naor, and Yuval Peres.
Rigorous location of phase transitions in hard optimization problems.
Nature, 435:759–764, 2005.

Scott D. Stoller, Ping Yang, C. R. Ramakrishnan, and Mikhail I. Gofman.
Efficient policy analysis for administrative role based access control.
In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 445–455, 2007.

Kathi Fisler, Shriram Krishnamurthi, Leo A. Meyerovich, and Michael Carl Tschantz.

Verification and change-impact analysis of access-control policies.
In *International Conference on Software Engineering*, 2005.