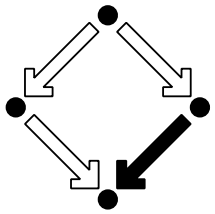
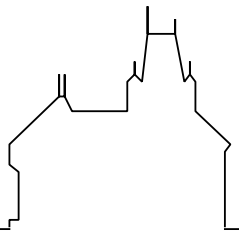


RISC-Linz
Research Institute for Symbolic Computation
Johannes Kepler University
A-4040 Linz, Austria, Europe



Sixth International Symposium on

Symbolic Computation in Software Science

SCSS 2014

Short Papers

December 7–8, 2014
Gammarth, Tunisia

Temur Kutsia and Andrei Voronkov
(Editors)

RISC Report Series No. 14-11

Sixth International Symposium on

SYMBOLIC COMPUTATION IN SOFTWARE SCIENCE

SCSS 2014

SHORT PAPERS

December 7–8, 2014
Gammarth, Tunisia

Preface

This collection contains short papers presented at the Sixth International Symposium on Symbolic Computation in Software Science, SCSS 2014, held on December 7–8, 2014, in Gammarth, Tunisia. It was organized by the Tunisian Society for Digital Security and the Research Unit of Digital Security, in collaboration with the Higher School of Communication of Tunis (University of Carthage), The Eos project, University of Tsukuba, The University of Manchester, and the Research Institute for Symbolic Computation of the Johannes Kepler University Linz.

SCSS 2014 solicited papers in two categories: regular and short papers. We received 8 submissions in the regular track, and 9 in the short papers track. After reviewing, the Program Committee selected 5 regular and 7 short papers. The symposium program also includes two invited talks: by Nikolaj Bjørner and William M. Farmer, and the invited tutorial by Stephen M. Watt.

The regular papers appeared in the EasyChair Proceeding in Computing. The short papers are published in this volume. The submission, Program Committee work, and preparation of the symposium program and proceedings were organized through the EasyChair system.

We would like to thank the Program Committee members and reviewers for their efforts. Thanks are also due to the Conference Chairs Adel Bouhoula and Tetsuo Ida and the Organization Committee chair Mohamed-Bécha Kaâniche for their work in the preparation and organization of the symposium.

December 2014

Temur Kutsia
Andrei Voronkov

Program Committee

Elvira Albert	Complutense University of Madrid
Adel Bouhoula	Higher School of Communications of Tunis
James H. Davenport	University of Bath
Roberto Giacobazzi	University of Verona
Arie Gurfinkel	Software Engineering Institute, Carnegie Mellon University
Nao Hirokawa	JAIST
Tetsuo Ida	University of Tsukuba
Florent Jacquemard	INRIA - IRCAM
Laura Kovacs	Chalmers University of Technology
Temur Kutsia	RISC, Johannes Kepler University Linz
Ali Mili	NJIT
Joel Ouaknine	Department of Computer Science, Oxford University
Ruzica Piskac	Yale University
Andrei Voronkov	University of Manchester
Dongming Wang	Beihang University and UPMC-CNRS

Table of Contents

A Library of Anti-Unification Algorithms*

Alexander Baumgartner and Temur Kutsia

RISC, Johannes Kepler University, Linz, Austria

Abstract

Generalization problems arise in many areas of software science: code clone detection, program reuse, partial evaluation, program synthesis, invariant generation, etc. Anti-unification is a technique used often to solve generalization problems. In this paper we describe an open-source library of some newly developed anti-unification algorithms in various theories: for first- and second-order unranked terms, higher-order patterns, and nominal terms.

1 Introduction

Given two terms t_1 and t_2. The anti-unification problem is concerned with finding a generalization term t such that both, t_1 and t_2 are instances of t under some substitution. Interesting generalizations are the least general ones. In software science such problems arise, for instance, in software code clone detection [6, 10], program verification [11], program synthesis [13], partial evaluation [1, 8], invariant generation [6], etc.

The open-source library described in this paper implements anti-unification for unranked terms, higher-order patterns, and nominal terms. Generalization problems in these theories may arise, for instance, in proof generalization or analogical reasoning in higher-order or nominal logic, in learning or refactoring λ-Prolog and α-Prolog programs, in detection of similarities in XML documents or in pieces of software code, just to name a few. Therefore, the algorithms provided by the library can be a valuable ingredient for tools that need to solve such generalization problems.

To be more specific, the library contains implementations of

- first-order rigid unranked anti-unification from [9],
- second-order unranked anti-unification from [3],
- higher-order anti-unification from [4] (and its subalgorithm for deciding α-equivalence),
- nominal anti-unification from [5] (and its subalgorithm for deciding equivariance).

It consists of four Java libraries for four anti-unification algorithms (urau.jar, urauc.jar, hoau.jar and nau.jar), which have the same structure. There is one main package which starts with the name `at.jku.risc.stout`, followed by a short abbreviation for the implemented algorithm (e.g. `urau`, `urauc`, `hoau` or `nau`). It contains three subpackages, namely `algo`, `data` and `util`.

The package `algo` contains the algorithmic part, for instance a Java class named `AntiUnify` which serves as entry point of the respective anti-unification algorithm. Java classes which represent a specific data structure like a term or an equation system are located in the `data` package. This package also offers a default implementation of an input parser, named `InputParser`. The `util` package holds some utility classes like `DataStructureFactory` which is used by the library to instantiate common structures (e.g., lists, queues, maps, sets). The user of the library is free to choose an arbitrary implementation of those data structures.

Each of the implemented algorithms has a separate Web page with a convenient Web interface to try it online. There are also the link to the paper where the algorithm is described, a brief explanation of the syntax, and some examples. Besides using the Web interface, the user may try also a shell version

*A full version of this system description [2] appeared in the proceedings of the 14th European Conference on Logics in Artificial Intelligence (JELIA 2014). This short paper is a summary of the original work.

of each algorithm, or download the sources, or embed the algorithm in her/his own project. A sample code of the latter option is also available from the Web.

In this paper, for each algorithm mentioned above we define the problem it solves, give a simple example, indicate its Web address, and explain how it can be embedded in users projects.

2 Unranked First-Order Anti-Unification

The problem of unranked anti-unification is formulated for terms defined over unranked alphabet. Hedge variables are used to fill in gaps in generalizations, while term variables abstract single subterms with different top function symbols. Unranked anti-unification is finitary, but it turned out that a minimal and complete algorithm may compute up to 3^n generalizations, where n is the input size. Therefore, the notion of $\mathscr{R}_{\mathsf{T}}$-*generalization* has been introduced in [9].

Definitions. Given pairwise disjoint countable sets of unranked function symbols $\mathscr{F}$ (symbols without fixed arity), term variables $\mathscr{V}_{\mathsf{T}}$, and hedge variables $\mathscr{V}_{\mathsf{H}}$, the following grammars define *terms* $t ::= x \mid f(\tilde{s})$, *hedge elements* $s ::= t \mid X$, and hedges $\tilde{s} ::= s_1, \ldots, s_n$, where $x \in \mathscr{V}_{\mathsf{T}}$, $f \in \mathscr{F}$, $X \in \mathscr{V}_{\mathsf{H}}$, and $n \geq 0$.

Given two hedges $\tilde{s}$ and $\tilde{q}$, an *alignment* is a sequence of the form $f_1\langle I_1, J_1\rangle \ldots f_m\langle I_m, J_m\rangle$ such that $I_1 < \cdots < I_m$, $J_1 < \cdots < J_m$, and f_k is the symbol at position I_k in $\tilde{s}$ and at position J_k in $\tilde{q}$ for all $1 \leq k \leq m$. With $<$ we denote the (strict) lexicographic ordering on positions.

A *rigidity function* $\mathscr{R}$ is a function that returns a set of alignments for two hedges with all the positions in the alignments being singleton integers (allowing only top symbols). Typical examples of rigidity functions are those which return longest common subsequences or longest common substrings of the top symbols of the input hedges.

The implemented anti-unification algorithm solves the following problem:
Given: Two variable-disjoint hedges $\tilde{s}$ and $\tilde{q}$ and the rigidity function $\mathscr{R}$.
Find: A complete set of $\mathscr{R}_{\mathsf{T}}$-generalizations for $\tilde{s}, \tilde{q}$ and $\mathscr{R}$.

For instance, $\{(g(a,a), X, f(g(a), g(Y))), (X, g(x,x), f(g(a), g(Z)))\}$ is the minimal complete set of $\mathscr{R}_{\mathsf{T}}$-generalization, of the hedges $(g(a,a), g(b,b), f(g(a), g(a)))$ and $(g(a,a), f(g(a), g))$, where $\mathscr{R}$ computes longest common subsequences.

How to use. We assume that there are two data sources `in1` and `in2` available in form of `Reader` instances, each of them containing one of the hedges to be generalized. Moreover, the variable `eqSys` is of appropriate type. We explain the usage of the library on a code fragment:

```
1  RigidityFnc rFnc = new RigidityFncSubsequence();
2  eqSys = new EquationSystem<AntiUnifyProblem>() {
3    public AntiUnifyProblem newEquation() {
4      return new AntiUnifyProblem();
5    } };
6  new InputParser<>(eqSys).parseHedgeEquation(in1, in2);
7  new AntiUnify(rFnc, eqSys, DebugLevel.SILENT) {
8    public void callback(AntiUnifySystem res, Variable var) {
9      System.out.println(res.getSigma().get(var));
10   }; }.antiUnify(true, null);
```

There are two rigidity functions available from the library. The one which is used in the first line of the code fragment computes longest common subsequence alignments. The other one is called `RigidityFncSubstring` and computes longest common substring alignments. It is easy to implement a different rigidity function. One simply has to extend the base class `RigidityFnc` which is provided by the library.

The lines 2 to 5 show the instantiation of an equation system which is of type `AntiUnifyProblem`. It is used in line 6 to instantiate a parser instance. In the same line, the input sources are used to create one equation of two hedges, which is added to `eqSys`. One could add more equations to the system by just calling the method `parseHedgeEquation(in3, in4)` again.

After specifying the rigidity function and parsing the equation system, the main algorithm `AntiUnify` is invoked using this data (line 7). For production use we want to silently compute all the generalizations and process them by a callback function, which is defined in the lines 8 to 10. The callback function, which is invoked for each generalization, provides two arguments for the implementation. The first one is of type `AntiUnifySystem` and contains all the data which has been collected during the run: The substitution `getSigma`, the store `getStore` and some additional information. The second argument is the generalization variable. Line 9 prints the computed generalization, which is the value associated with the variable in the substitution.

During the anti-unification process, fresh variables are introduced. They are named by a sequence number which is put between a prefix and a suffix. The counter for generating the number sequence can be reset by calling the function `NodeFactory.resetCounter`. The prefix and suffix for fresh variables can also be specified by static variables of the class `NodeFactory`.

Web page. `http://www.risc.jku.at/projects/stout/software/urau.php`.

3 Unranked Second-Order Anti-Unification

The language used in section 2 does not permit higher-order variables. This imposes a natural restriction on solutions: The computed lggs do not reflect similarities between input hedges, which are located under distinct heads or at different depths. For instance, $f(a,b)$ and $g(h(a,b))$ are generalized by a single variable, although both terms contain a and b and a more natural generalization could be, e.g., $\mathring{X}(a,b)$, where $\mathring{X}$ is a higher-order variable. In applications, it is often desirable to detect these similarities. Therefore, in [3], an anti-unification algorithm has been developed where second-order power is gained by using context variables.

Definitions. Given pairwise disjoint countable sets of unranked function symbols $\mathscr{F}$, hedge variables $\mathscr{V}_{\mathsf{H}}$, unranked context variables $\mathscr{V}_{\mathsf{C}}$, and a special symbol $\circ$ (the hole), the following grammars define *terms* $t ::= X \mid f(\tilde{s}) \mid \mathring{X}(\tilde{s})$, *hedges* $\tilde{s} ::= t_1,\ldots,t_n$, and *contexts* $\tilde{c} ::= \tilde{s}_1,\circ,\tilde{s}_2 \mid \tilde{s}_1,f(\tilde{c}),\tilde{s}_2 \mid \tilde{s}_1,\mathring{X}(\tilde{c}),\tilde{s}_2$, where $X \in \mathscr{V}_{\mathsf{H}}$, $f \in \mathscr{F}$, $\mathring{X} \in \mathscr{V}_{\mathsf{C}}$, and $n \geq 0$.

We only give an informal definition of *admissible alignments*: An alignment $\mathfrak{a}$ of two hedges $\tilde{s}$ and $\tilde{q}$ is called *admissible* iff there exists a generalization $\tilde{g}$ of $\tilde{s}$ and $\tilde{q}$ which contains all the corresponding symbols from $\mathfrak{a}$. We call $\tilde{g}$ a *supporting generalization* of $\tilde{s}$ and $\tilde{q}$ with respect to $\mathfrak{a}$.

Least general supporting generalizations might not be unique. For instance, for (a,b,a) and (b,c) with the admissible alignment $b\langle 2,1\rangle$, we have two supporting least general generalizations (X,b,X,Y) and (X,b,Y,X). Therefore, we are interested in a special class of supporting generalizations, which we call $\mathscr{R}_{\mathsf{C}}$-generalizations. It guarantees uniqueness of the result.

The implemented anti-unification algorithm has $O(n^2)$ time complexity and $O(n)$ space complexity, where n is the size of the input. It solves the following problem:
Given: Two variable-disjoint hedges $\tilde{s}$ and $\tilde{q}$ and their admissible alignment $\mathfrak{a}$.
Find: A least general $\mathscr{R}_{\mathsf{C}}$-generalization of $\tilde{s}$ and $\tilde{q}$ with respect to $\mathfrak{a}$.

For instance, $\mathring{X}(a,b)$ is an $\mathscr{R}_{\mathsf{C}}$-generalization of $f(g(a,b,c))$ and (a,b) with respect to $a\langle 1{\cdot}1{\cdot}1,1\rangle$ $b\langle 1{\cdot}1{\cdot}2,2\rangle$, while $\mathring{X}(a,b,X)$ and $\mathring{X}(\mathring{Y}(a,b))$ are not.

How to use. The usage of this algorithm is very similar to the one we explained in section 2. Instead of a rigidity function there is an alignment computation function. The library offers two such functions: The first one, called `AlignFncLAA`, computes longest admissible alignments.

The other one is `AlignFncInput` and can be used to specify a certain admissible alignment. The admissibility test for this alignment has to be done in advance. Therefore the `Alignment`-class offers a method `isAdmissible` which returns `true` iff an alignment is admissible. Alignment computation functions have the common base class `AlignFnc`. This base class can be used to implement other alignment computation functions.

Web page. `http://www.risc.jku.at/projects/stout/software/urauc.php`.

4 Higher-Order Pattern Anti-Unification

The higher-order anti-unification algorithm described in [4] works on simply typed λ-terms: It takes as input two such terms of the same type, in η-long β-normal form, and returns their least general pattern generalization. Patterns here mean higher-order patterns à la Miller [12]. (Note that it is not required the input to be patterns.) Such a generalization always exists, is unique modulo α-equivalence and variable renaming, and can be computed in cubic time within linear space with respect to the size of the input, see [4].

Definitions. Simple types are constructed from *basic types* δ with the help of the type constructor $\rightarrow$ by the grammar $\tau := \delta \mid \tau \rightarrow \tau$. *Variables* and *constants* have an assigned type. Then λ*-terms* t are built using the grammar: $t ::= x \mid c \mid \lambda x.t \mid (t_1\ t_2)$, where x is a typed variable and c is a typed constant. Terms like $(\ldots(h\ t_1)\ldots t_m)$, where h is a constant or a variable, are written as $h(t_1,\ldots,t_m)$.

A *higher-order pattern (HOP)* is a λ-term, in which, when written in η-long β-normal form, all free variables apply to pairwise distinct bound variables.

Given two variable-disjoint λ-terms t_1 and t_2, we say that a λ-term t that generalizes both t_1 and t_2 is their *higher-order pattern generalization*, if t is an HOP.

The HOP anti-unification (HOPAU) algorithm solves the following problem:
Given: Higher-order terms t_1 and t_2 of the same type in η-long β-normal form.
Find: A least general higher-order pattern generalization of t_1 and t_2.

For instance, if $t_1 = \lambda x,y.f(h(x,x,y),h(x,y,y))$ and $t_2 = \lambda x,y.f(g(x,x,y),g(x,y,y))$, then the term $t = \lambda x,y.f(X(x,y),Y(x,y))$ is a higher-order pattern lgg of t_1 and t_2.

How to use. The usage of this algorithm is even easier to the one we explained in section 2, because there is no need to define a rigidity function. Otherwise it is very similar.

Web page. `http://www.risc.jku.at/projects/stout/software/hoau.php`.

5 Nominal Anti-Unification

Nominal techniques [7] have been introduced to formally represent and study systems with binding. The nominal anti-unification (NAU) algorithm developed in [5] takes as input two terms-in-contexts (pairs of a freshness constraint and a nominal term) and tries to compute a generalization term-in-context. Under the assumption that the set of atoms permitted in generalizations is finite, there is a unique lgg modulo variable renaming and α-equivalence. The algorithm has $O(n^4)$ time complexity and $O(n^2)$ space complexity, where n is the input size.

Definitions. Nominal terms contain *variables* $(X, Y, \ldots)$, *atoms* $(a, b, \ldots)$ and function symbols $(f, g, \ldots)$. Variables can be instantiated and atoms can be bound. A *swapping* $(a\,b)$ is a pair of atoms of the same sort. A *permutation* π is a sequence of swappings. It can apply to terms and cause swapping the names of atoms. *Nominal terms* t are given by the following grammar, where $a.t$ is abstraction and $\pi{\cdot}X$ is called suspension: $t ::= f(t_1, \ldots, t_n) \mid a \mid a.t \mid \pi{\cdot}X$. A suspension $\pi{\cdot}X$ postpones the application of a permutation π to X until X is instantiated. Substitution application allows atom capture, for instance, $a.X\{X \mapsto a\} = a.a$.

A *freshness context* ∇ is a finite set of pairs of the form $a\#X$ stating that the instantiation of X cannot contain free occurrences of a. A *term-in-context* is a pair $\langle \nabla, t \rangle$ of a freshness context ∇ and a term t. A term-in-context $\langle \nabla, t \rangle$ is *based* on a set of atoms A, if all the atoms which occur in t and ∇ are elements of A.

The NAU algorithm solves the following problem:

Given: Two nominal terms t_1 and t_2 of the same sort, a freshness context ∇, and a *finite* set of atoms A such that $\langle \nabla, t_1 \rangle$ and $\langle \nabla, t_2 \rangle$ are based on A.

Find: A term-in-context $\langle \Gamma, t \rangle$ which is also based on A, such that $\langle \Gamma, t \rangle$ is a least general generalization of $\langle \nabla, t_1 \rangle$ and $\langle \nabla, t_2 \rangle$.

For instance, for $t_1 = f(b, a)$, $t_2 = f(X, (a\,b){\cdot}X)$, $\nabla = \{b\#X\}$, and $A = \{a, b\}$, the NAU algorithm computes the lgg of $\langle \nabla, t_1 \rangle$ and $\langle \nabla, t_2 \rangle$, which is $\langle \emptyset, f(Y, (a\,b){\cdot}Y) \rangle$.

How to use. To explain the library usage on a code example, we again assume the existence of two `Reader` instances `in1` and `in2` which contain the nominal terms to be generalized. Furthermore, we assume that there is a `Reader` instance `inA` for reading a set of atoms (e.g. `{c,d,...}`) and `inN` for the freshness context (e.g. `{a#X,b#Y,...}`).

```
1  final NodeFactory factory = new NodeFactory();
2  eqSys = new EquationSystem<AntiUnifyProblem>() {
3    public AntiUnifyProblem newEquation(NominalTerm t,NominalTerm s) {
4      return new AntiUnifyProblem(t, s, factory);
5    } };
6  FreshnessCtx nablaIn = new InputParser(factory)
7      .parseEquationAndCtx(in1, in2, inA, inN, eqSys);
8  new AntiUnify(eqSys, nablaIn, DebugLevel.SILENT, factory) {
9    public void callback(AntiUnifySystem res, Variable var) {
10     System.out.println(res.getSigma().get(var));
11     System.out.println(res.getNablaGen());
12   }; }.antiUnify(false, null);
```

In contrast to the other libraries, an instance of `NodeFactory` is needed, which we create in line 1. The lines 2 to 5 demonstrate the creation of an equation system.

All the input sources are parsed in line 7. The new equation is added to `eqSys` and the parsed freshness context is returned. Moreover, the factory instance remembers all the parsed atoms regardless of the input source they come from. More equations may be added to `eqSys` by calling the method `parseEquation(in3, in4, eqSys)` from `InputParser`.

Line 11 shows that, additionally to the substitution and store, the generated freshness context is provided by the instance `res` of the class `AntiUnifySystem`.

Web page. `http://www.risc.jku.at/projects/stout/software/nau.php`.

Acknowledgments

Supported by the Austrian Science Fund (FWF) under the project SToUT (P 24087-N18).

References

[1] M. Alpuente, S. Escobar, J. Meseguer, and J. Espert. A modular order-sorted equational generalization algorithm. *Information and Computation*, 235:98–136, 2014.

[2] A. Baumgartner and T. Kutsia. A library of anti-unification algorithms. In E. Fermé and J. Leite, editors, *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, volume 8761 of *Lecture Notes in Computer Science*, pages 543–557. Springer, 2014.

[3] A. Baumgartner and T. Kutsia. Unranked Second-Order Anti-Unification. In U. Kohlenbach, editor, *Proceedings of the 21st Workshop on Logic, Language, Information and Computation, WoLLIC 2014* , volume 8652 of *Lecture Notes in Computer Science*, pages 66– 80. Springer, 2014.

[4] A. Baumgartner, T. Kutsia, J. Levy, and M. Villaret. A variant of higher-order anti-unification. In F. van Raamsdonk, editor, *RTA*, volume 21 of *LIPIcs*, pages 113–127. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[5] A. Baumgartner, T. Kutsia, J. Levy, and M. Villaret. Nominal anti-unification. In T. Kutsia and C. Ringeissen, editors, *Proceedings of the 28th International Workshop on Unification, UNIF 2014*, number 14-06, pages 62–68, 2014.

[6] P. E. Bulychev, E. V. Kostylev, and V. A. Zakharov. Anti-unification algorithms and their applications in program analysis. In A. Pnueli, I. Virbitskaite, and A. Voronkov, editors, *Ershov Memorial Conference*, volume 5947 of *Lecture Notes in Computer Science*, pages 413–423. Springer, 2009.

[7] M. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3-5):341–363, 2002.

[8] J. P. Gallagher. Tutorial on specialisation of logic programs. In D. A. Schmidt, editor, *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'93, Copenhagen, Denmark, June 14-16, 1993*, pages 88–98. ACM, 1993.

[9] T. Kutsia, J. Levy, and M. Villaret. Anti-unification for unranked terms and hedges. *J. Autom. Reasoning*, 52(2):155–190, 2014.

[10] H. Li and S. J. Thompson. Similar code detection and elimination for Erlang programs. In M. Carro and R. Peña, editors, *PADL*, volume 5937 of *Lecture Notes in Computer Science*, pages 104–118. Springer, 2010.

[11] J. Lu, J. Mylopoulos, M. Harao, and M. Hagiya. Higher order generalization and its application in program verification. *Ann. Math. Artif. Intell.*, 28(1-4):107–126, 2000.

[12] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Log. Comput.*, 1(4):497–536, 1991.

[13] U. Schmid. *Inductive Synthesis of Functional Programs, Universal Planning, Folding of Finite Programs, and Schema Abstraction by Analogical Reasoning*, volume 2654 of *Lecture Notes in Computer Science*. Springer, 2003.

A Self-Disciplined Privacy Oriented Access Control Framework for Public Clouds

Maherzia Belaazi, Hanen Boussi Rahmouni and Adel Bouhoula

Higher School of Communication of Tunis, University of Carthage, Tunisia.
{maherzia.belaazi,hanen.boussi,adel.bouhoula}@supcom.tn

Abstract

While the transformative power of cloud computing in terms of cost scalability and agility is widely known, one of the first questions that arises is how the data hosted in the cloud-is accessed, stored and used? In this context, privacy preserving is a key user concern when judging the adoption of clouds in domains where sensitive personal information are highly involved. Indeed, as far as cloud providers are concerned, they should ensure the privacy protection of data hosted in clouds on behalf of their customers. Equally, they should satisfy the needs of law enforcement. It is a fact that access control is one of the essential and traditional security mechanisms of data protection. However, in the context of open and dynamic environments such as clouds, access control becomes more complicated. This is because the security policies, models and related mechanisms have to be defined across heterogeneous security domains. Thus, improving the current access control paradigms is crucial in order to ensure privacy compliance in distributed environments. In this paper, we aim to produce a self-disciplined access control framework for public clouds. We believe that a formal knowledge representation of access control policies, in particular, when these policies are able to integrate data protection requirements, could enforce privacy compliance at runtime. Indeed, we mainly focus on ontologies, as formalized conceptual models for access rules expression that conducts a private and secure reasoning. It could resolve issues like ambiguity in the expression of privacy legislation. The issue of the interoperability between various jurisdictions, denoted by the geographical area of the different cloud actors, could also be addressed in a formal manner.

1 Introduction : Privacy and Access Control standards

Cloud providers should ensure the privacy protection of data hosted in the cloud on behalf of customers (Pearson, 2013) and they should satisfy the needs of law enforcement (Vimercati, 2010). For a large distributed system like a cloud system, access decision needs to be more flexible and scalable (Khan, 2012). The problems that need to be considered are principally two: first, the definition of privacy integrating access control policies. This requires considering highly expressive specification languages and solutions for combining data protection requirements. Second, some information may not be under the control of a single authority (Yang, 2013). These multi-authorities' scenarios should be supported from the administration point of view providing solutions for modular, large-scale, scalable policy composition and interaction (Damiani, 2005). A lot of work has been done in the area of security policy languages. Many of these approaches have mainly captured the access control aspect of privacy compliance. Since we are in particular looking for formal representation and design, we choose to study some of XML based privacy access control policy languages: P3P (P3P, 2007), EPAL (Powers, 2004) and XACML (XACML, 2013). These privacy policy languages (P3P, EPAL and XACML) are formal languages that are specifically designed to facilitate the expression of privacy policies, practices and requirements. These languages have various advantages:

- A potential for automatic policy enforcement of dataaccess, use and storage limitation requirements.

- A standardization level for automatic policy evaluation, something which is not possible with expressing policies in human language

But in a context of open dynamic heterogeneous environment (in public cloud: heterogeneous data center resources are shared by a vast number of users with diverse privacy obligations) these languages must be improved by:

- Explicitly mapping access control requirements (cloud users conditions and cloud providers rules) with privacy requirements (owner preferences and laws obligations).
- Examining according to regulations data lifecycle: from access, use to deletion.
- Explicitly incorporating references to the legislation, while expressing access control policies. In our scope, we mean by the reference to legislation: the original text-law, the entity (country or state or united states) proposing this law and the legal strength of such law. The text law expresses the law articles used for an access control policy. These legal articles belongs to different type of legislations (national or international, acts of parliament or orders, .) that have different power and priority which we refer to as ”legal strength”.

2 Requirements for a Legislation Driven Framework

With the emergence of cloud systems and due to their economic benefits, many organisations become more interested in using them. These organisations usually act as either a direct user of the cloud system or as both a user and a participant at the same time. When participating as a cloud member, an organisation needs to share some resources with other members which could be either a web service or a collection of data. The access to the shared resource will be, primarily managed by the cloud access control unit and would be a consequence of evaluating some security policies. However in many cases the cloud members would like to have some autonomy with regards to controlling their shared resources. It is therefore required that the cloud system could consult the service or data provider and allow them to provide additional policies applicable to the usage of their resources. The provided policies will have to be combined with another set of clouds standard policies identified by the cloud access control unit. Besides, with growing pressure of the need to meet privacy regulations (Pearson, 2013)(Khan, 2012). These policies must ensure for not only for access and usage of the data but also for compliance with privacy and other regulations in question. In summary, three policy categories should be examined: provider's policies, user's policies and legal policies. For example, the patient's data flow in a cloud computing solution is a typical case (figure 1). Let's now have a look at this scenario: A patient takes some medical tests to help diagnosing medical problem. When completed the tests results will be hosted in a cloud solution. The patient prefers to stay anonymous and not sharing the result of their tests with any person (a data owner policy example). The cloud provider standard policies permit the collecting of medical information for specific research purposes (a cloud provider policy example). The Doctor could ask to share results with other doctor to get more advice (an example of data processing/access control request). In some territories' legislation, a personal data processing (collecting or sharing) should get the consent of the data owner (An example of legislative policy), and other legislation force data sharing in some cases of national threat (like contagious disease).... Over the previous scenarios we describe how many policy categories could be involved in a same cloud computing scenarios.

Hence, in the previous cloud scenarios we have to deal with the problem of integrating heterogeneous policies. The differences between clouds standard policies, service providers policies and legislative policies could be identified on many grounds. This could include syntactic differences, semantic differences (either linguistic or conceptual) or the priority of policy evaluation at runtime.

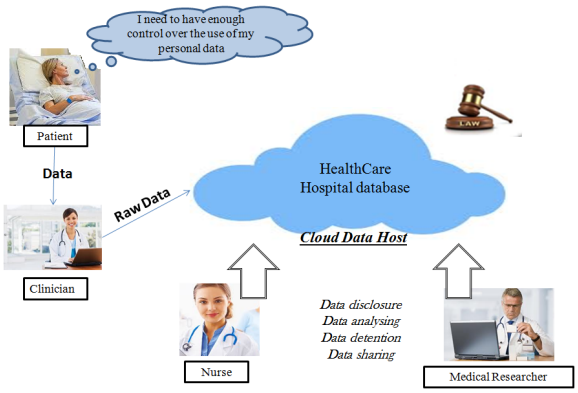


Figure 1: Medical Data Flow in cloud computing solution.

For syntactic differences: Improvement and extension of standard access control model and languages is a required solution. XACML is our target language since it was approved as an OASIS Standard and seems appropriate for future research and standards efforts related to privacy policy languages (Anderson, 2005). Hence, we choose to extend XACML with privacy requirements in addition to possible delegation of access control definition and evaluation. Personal data protection rules defined by law (Pearson, 2013) should be taken into consideration.
For semantic differences: we look to enforce access control policies which we have specified through some added semantics provided by the power of ontology languages and tools. We believe it would promote common understanding among participants and would ensure greater interoperability between cloud security domains.
For priority evaluation: During policies evaluation, the case of conflicts between different applicable policies is a likely case. In this stage, we believe that the reference to legislative priority (text-law, act, etc..) should in some parts drive the decision making. For example in the UK common law has higher priority of enforcement compared with other primary law (e.g. medical, data protection law).

3 Overview of the Framework

The generalized policy management architecture (figure 2), suggested by the IETF (Internet Engineering Task Force) policy architecture draft [IETF Policy Framework Working Group 2003] is being used by commercial vendors as the basis of designing policy architectures. It includes a policy management service, a dedicated policy repository, at least one policy decision point (PDP) and at least one policy enforcement point (PEP). The PDP embodies the decision-making functionality of policy-based management. There are one or more such policy servers in a control domain, with each server configured to support policy management for some defined group of policy clients or PEPs in the domain.

We propose to improve the previous architecture by adding a ”privacy protection point” (figure3). This point will ensure the compliance to privacy requirements dictated by the provider and the user. It will also ensure compliance with legislation requirements.

In order to achieve our goal, we will proceed -as described in figure 4- in the following steps steps:

1. Analyse access control meta-models requirements and concepts related to open and dynamic environments. Here, we will focus on some existingaccess control models including attribute based access control (Priebe, 2006), Contextual access control (Covington, 2006) and Usage oriented

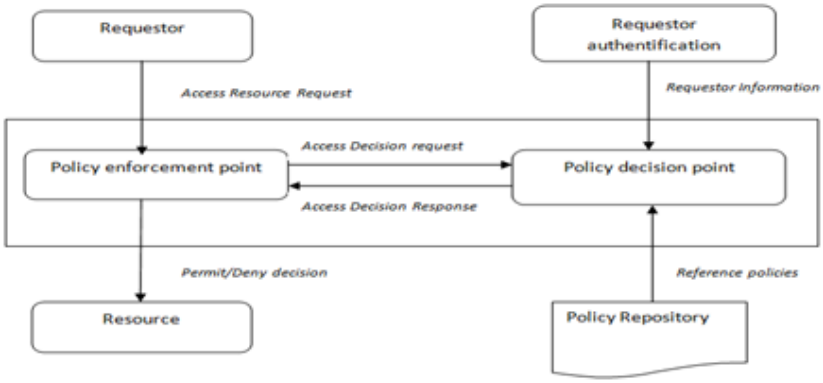


Figure 2: Access control and Policy management architecture.

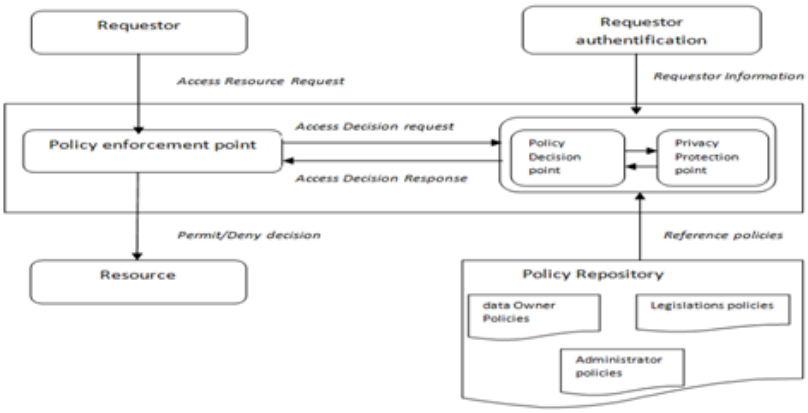


Figure 3: Policy management architecture enforced by privacy protection point.

access control (Park, 2007). We will also analyse models that have proposed a privacy profile. The Purpose access control model (Byun, 2005) is a good example. As first preliminary analysis, these models dont refer to legislation issues and doesn't deal with possible delegation of access control between users. So a new access model that is legislation based should be proposed.

2. Analyse legislation requirements for privacy protection in order to clarify the ambiguity of the reference text law behind it. (Pearsan, 2013).

3. Define a formal conceptual legislation driven model by proposing an ontology for fine-grained access control requirements. This ontology would be extended by adding privacy and data protection obligations extracted from legislation. At a later stage this semantic formal ontology will be a base for an inference system.

In our scope, the privacy protection point described previously in (figure 3) is the same Inference engine produced in (figure4). This reasoning engine will help in making a self-disciplined access decision in distributed heterogeneous environments; It will enforce the "policy decision point" in order to verify if an access decision is legitimate. And in the case of jurisdictional conflicts between countries, it could be a good guide in order to decide which policy to apply. As a solution, we propose the use

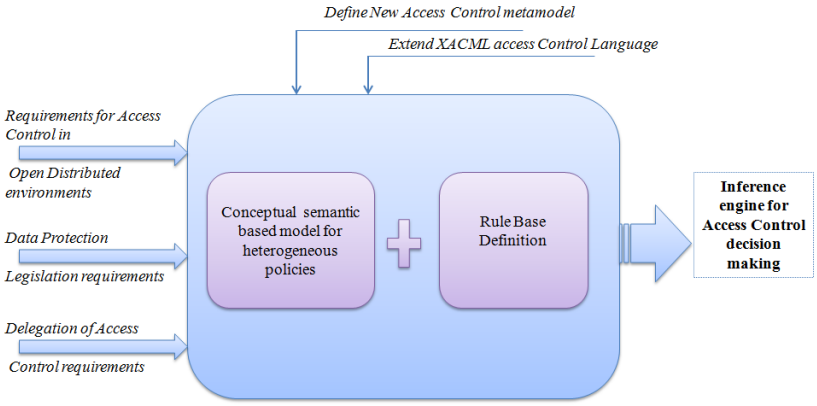


Figure 4: Towards a self-disciplined Access Control decision making.

of references to text law and its legal strength as parameters to be formally presented while expressing access control policies. Based on the semantic web language, the proposed engine should help in concluding about semantic matching of terms across different sites involved in cloud scenarios. Indeed, it ensures that the entity ”requestor” and the entity ”provider” of an access control policy context shares the same meaning of the involved entities or describes equivalent attributes. (For example, ”doctor” or ”practitioner” should refer to the same subject). Our ”privacy protection point” must take in consideration the possibility of access control delegation between the cloud provider and its clients (the cloud service requestor). This federation of access control could also take place between users. For example, for unavailability reasons, a user in an organisation could delegate some of its privileges to another user. That should be carefully expressed in an access control policy.

4 Conclusion

In an attempt to urge public cloud trust, we propose to enforce access controls by preserving data protection and privacy requirements via a framework that is driven by legislation. In our research, we focus on the requirements for sensitive data protection driven by legislation. We look on how to incorporate these requirements in an access control model while expressing security policies for an open and dynamic environment such as the cloud. From the literature we could state two recent works that deal with access control issues in cloud computing. The first one (Reul, 2013) proposes an ontology-based access control covering context, task and role-based models. The second (Choi, 2013), proposes an ontology that covers core elements of security policies. This work is considering ontologies as a solution to semantic interoperability when evaluating access control policies in distributed environments. We think that both works don’t deal with privacy compliance requirements especially the requirements defined by legislation in the field of personal data protection. Also, these works don’t deal with the need for access control delegation in the cloud. Our framework aims to ensure an access control mechanism that incorporates three levels of policies: the data owner’s policies, the cloud provider’s policies and the legislation based policies. In order to achieve this purpose, we take advantage of semantic web technologies: a growing technology allowing policies to be richly described over heterogeneous domain data. Besides, it promotes a common understanding among organizations. We believe that this will help in building comprehensive and verifiable security properties for open, dynamic environments such as public clouds since it requires interoperability across multiple organizations with additional level of

ambiguous and demanding compliance with legislation. In this paper, we have described our research proposal and our driven research methodology. Finally, it is worth noting that this work is at an early stage of the required investigation according to the research methodology described in a previous section.

References

S. Pearson and G. Yee. (2013). Privacy and Security for Cloud Computing, Computer Communications and Networks,Springer-Verlag London.

A. Raouf Khan, (2012). Access control in cloud computing environment. Asian Research Publishing Network (ARPN) Journal of Engineering and Applied Science.

E. Damiani ,S De Capitani di Vimercati, P. Samarati, (2005). New Paradigms for Access Control in Open Environments. Signal Processing and Information Technology. Proceedings of the Fifth IEEE International Symposium.

P3P, (2007). Platform for Privacy Preferences (P3P) Project. Retrieved from http://www.w3.org/P3P/.

Powers, C., Adler, S. Wishart, B., (2004). EPAL Translation of the Freedom of Information and Protection of Privacy Act. White Paper. IBM Tivoli and Information and Privacy Commissioner/Ontario.

XACML, (2013). eXtensible Access Control Markup Language (XACML) Version 3.0 Retrieved from http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

Anne h. Anderson, (2005). A Comparison of Two Privacy Policy Languages: EPAL and XACML. Sun Microsystems Labs Technical Report, November.

Priebe, T., Dobmeier, W. Kamprath, N.. (2006). Supporting Attribute-based Access Control with Ontologies. In Proceedings of the First International Conference on Availability, Reliability and Security., 2006. IEEE Computer Society.

Michael J. Covington and Manoj R. Sastry, (2006). A Contextual Attribute-Based Access Control Model. On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops.

J. Park and R. Sandhu., (2007). The UCONABC usage control model. ACM Transactions on Information and System Security.

JiWon Byun, Elisa Bertino, Ninghui Li., (2005). Purpose Based Access Control of Complex Data for Privacy Protection. Proceedings of the tenth ACM symposium on Access control models and technologies, Pages 102 - 110, ACM New York, NY, USA.

Quentin Reul, Gang Zhao, Robert Meersman, (2013). Ontology-based Access Control Policy Interoperability. STARLab 2013.

Chang Choi, Junho Choi, Pankoo Kim, (2013). Ontology-based access control model for security policy reasoning in cloud computing. Springer Science+Business Media New York 2013

Work-In-Progress: Repairing a Loop by Constructive Transformation using Mutation Analysis

Nafi Diallo[1] and Wided Ghardallou[2]

[1] CCS, NJIT, Newark New Jersey, U.S.A.
nafi.c.diallo@njit.edu
[2] Faculy of Sciences of Tunis, Tunis El Manar, Tunisia
wided.ghardallou@gmail.com

Abstract

One of the issues with traditional mutation testing is the possible large number of mutants generated. In this work, we illustrate how the concept of relative correctness, defined in [6] can guide the generation and selection of mutants when repairing a loop program. We show that fewer and better mutants can be generated , thus adding efficiency by reducing the huge computational cost associated with mutation analysis.

1 Introduction

In traditional mutation testing, the generation of mutants is achieved through source code inspection and mutant selection is tackled using testing. A problem that remains open is the computational cost associated with the large set of mutants that can be generated. Various methods such as selective mutation [3], mutant sampling [7] have been proposed to deal with this issue. And despite the tremendous effort put into the test data generation, mutation testing can still accepts or rejects a mutant for the wrong reason [1]. Trough the concept of Relative Correctness introduced in [6],we challenge both the generation of mutants and the selection of mutants. We undertake program repair by generating few mutants that are to be compatible with the specification, then by selecting mutants by verification rather than testing and finally by testing for relative correctness.

In the following sections, we present the concepts of invariant relation and relative correctness and describe how fault localization and program repair are achieved. We then present the results of their application to a sample program. Finally we conclude with the next steps.

2 Theoretical Foundations

3 Relational Definitions and Operations

In this section, we briefly present some relational notations and operations. We consider a set S defined by the values of some program variables, say x and y; we denote elements of S by s, and we note that s has the form $s =< x,y >$. We denote the x-component and (resp.) y-component of s by $x(s)$ and $y(s)$. We may use x to refer to $x(s)$ and x' to refer to $x(s')$. We refer to S as the space of the program and to $s \in S$ as a state of the program. A relation on set S is a subset of the Cartesian product $S \times S$. Constant relations on S include: the universal relation $L = S \times S$, the empty relation $\phi = \{\}$, and the identity relation denoted by I. Relations that have the form $R = C \times S$, for a subset C of S are called vectors. Operations on relations include the usual set theoretic operations of union ($R \cup R'$), intersection ($R \cap R'$), difference ($R \backslash R'$) and complement ($\overline{R} = L \backslash R$). It also includes the inverse of a relation defined as $\widehat{R} = \{(s,s') | (s',s) \in R\}$, the product of relations R and R' denoted by RR' and defined

by $RR' = \{(s,s')|\exists t : (s,t) \in R \wedge (t,s') \in R'\}$, the n^{th} power of relation R is the relation defined by $R^0 = I$, $R^{n+1} = RR^n$, the reflexive transitive closure of relation R is defined by $R^* = \{(s,s')|\exists n \geq 0 : (s',s) \in R^n\}$, the domain of relation R is the set defined as $dom(R) = \{s|\exists s' : (s,s') \in R\}$. We leave it to the reader to check that the vector that corresponds to the domain of a relation R is nothing but RL. The range of relation R is the domain of $\widehat{R}$. The pre-restriction of relation R to predicate t is the relation defined by $\{(s,s')|t(s) \wedge (s,s') \in R\}$. As for properties of relations, we say that relation R is reflexive if and only if $I \subseteq R$, and we say that R is transitive if and only if $RR \subseteq R$. We admit without proof that the reflexive transitive closure of relation R is the smallest superset of R that is reflexive and transitive. Also, we admit that R is a vector if and only if $RL = R$. We say that R is deterministic (or that it is a function) if and only if $\widehat{R}R \subseteq I$.

3.1 Program Semantics

Given a program p on space S, we let P be the function of p. P is defined by the set of pairs (s,s') such that if p starts execution on s, then it terminates in state s'. The domain of P (denoted by $dom(P)$) is the set of states s such that if p starts execution on s, then it terminates.
We consider while loops written in some C-like programming language. Our semantic definition of a while loop is due the following theorem introduced in [5]:

Theorem 1. *Let w be a while loop of the form `while(t)do{b}`. Then its function W is given by:*

$$W = (T \cap B)^* \cap \overline{\widehat{T}}$$

where B is the function of b and T is the vector defined by: $\{(s,s')|t(s)\}$.

3.2 Invariant Relation

Definition 1. *Let w be a while loop of the form `while(t)do{b}` on space S. B is the function of b and T is the vector defined by: $\{(s,s')|t(s)\}$. We say that relation R is an invariant relation for w if and only if it is a reflexive and transitive superset of $(T \cap B)$*

The interest of invariant relations is that they are approximations of $(T \cap B)^*$, the reflexive transitive closure of $(T \cap B)$; smaller invariant relations are better, because they represent tighter approximations of the reflexive transitive closure; the smallest invariant relation is $(T \cap B)^*$.

3.3 Relative Correctness

To define relative correctness, we introduce the concept of refinement.

Definition 2. *We let R and R' be two relations on space S. We say that R refines R' if and only if*

$$RL \cap R'L \cap (R \cup R') = R$$

We write this relation as: $R \sqsupseteq R'$ or $R' \sqsubseteq R$.

Following [6], relative correctness is defined as follows.

Definition 3. *Let R be a specification on space S and let g and g' be two programs on space S whose functions are respectively G and G'. We say that program g is more-correct than program g' with respect to specification R (abbreviated by: $G \sqsupseteq_R G'$) if and only if:*

$$(G \cap R)L \supseteq (G' \cap R)L$$

.
Similarly, we define the notion of strictly-more correct.A program g is strictly more-correct than program g with respect to specification R (abbreviated by: $G \sqsupset_R G'$) if and only if:

$$(G \cap R)L \supset (G' \cap R)L.$$

[6] describes details of these concepts and includes relevant examples.

4 Fault Localization and Mutant Generation

The approach for traditional mutant generation is inadequate because if fails to take into account the specification(s) of the program. Before applying mutation analysis, we argue that we can achieve a more efficient fault localization by using the notion of relative correctness as defined in [6]. Applying the method described in [4], we can localize the faults using the process of correctness verification. We do so by finding among the identified invariant relations, those that are incompatible with the specification. Our thesis is that we should only look at statements related to variables involved in these invariant relations as possible candidates for mutation. This allows us to significantly reduce the number of mutants to generate. Incidentally, we can say that we generate better mutants.

5 Program Repair and Mutant Selection

We argue that the selection of mutants in Traditional Mutation Testing is wrong because it uses absolute correctness. However, testing for absolute correctness is valid only if we assume that we are removing the last fault. We propose a program repair method that is a repetitive process in which we find new invariant relations that make them relatively more correct [6] than the original. The steps in the method are as follow:

- Step 1: Suggest a change
- Step 2: Find a new invariant relation, with the constraints that the status of the invariant relations identified so far must stay the same.
- Step 3: Check if the new invariant relation is compatible with the specification
- Step 4: Go back to Step 1 if there are more repairs to do, otherwise exit

6 Illustration

We apply the methods described above to the following loop g on space S defined by the variables $r,u,y,w,z,d,n,\ l,m,v,x,x0,t$ and b:

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
double r,u,y,w,z,d,n,l,m,v,p=0.25,k=0; // initial investment
y=w=v=0;
double x,x0 = 10000;
```

```
int t=0;
double a=.07;
double b;
b=a-0.01; //inflation adjusted rate
l=z=x=x0;
r=p;
d=n=m=0;
while ( r != p )
{
t= t + 1;n= n + x;
l=(1+b)*l; m= m + l;
k= k + 1000; y= n + k;
w= w + z;
z=(1+a) + z; v= w + k;
r= (v - y) / y; u= (m - n) / n;
d=r-u;
}}
```

Let R be the following specification on S:

$$R = \{(s,s')|w' == w - z \times \frac{(1-(1+a)^{(t'-t+1)})}{a} \wedge a \neq 0 \wedge x == x' \wedge k' - k == 1000 \times t'\}$$

When we deploy the algorithm outlined in [2] on this loop, we find the following invariant relations that are deemed compatible with the specification:

- $V_0 = \{(s,s')|t \leq t'\}$
- $V_1 = \{(s,s')|k \leq k'\}$
- $V_2 = \{(s,s')|k - 1000 \times t == k' - 1000 \times t'\}$
- $V_3 = \{(s,s')|x == x'\}$
- $V_4 = \{s,s'|1000 \times n' - k' \times x' == 1000 \times n - k \times x\}$
- $V_5 = \{s,s'|z' - (1+a) \times t' == z - (1+a) \times t\}$

We also find the following Q that is incompatible with the specification R:

$$Q = \{(s,s')|w - z\frac{(z-1-a)}{(2+2\times a)} == w' - z'\frac{(z'-1-a)}{(2+2\times a)}\}$$

Therefore the loop is not correct with respect to the specification R. We can infer that the variables that are mentioned in Q are the only variables that need to be changed in the program; also, they must be changed while preserving all the relevant invariant relations (i.e. the invariant relations that involve these variables). Using this condition, we derive the constraints that the identified variables must satisfy in their new form. For that purpose, we solve the following in Mathematica:

$$Constraints = \text{FullSimplify}\left[\text{Resolve}\left[\exists_{\{t,k,x,n,\text{tP},\text{kP},\text{xP},\text{nP},l,\text{lP}\}} V_0 \&\& V_1 \&\& .. \&\& V_5\right]\right]$$

We get:

$$(a+1 = 0 \wedge z = \text{zP}) \vee (a+1 < 0 \wedge \text{zP} \leq z) \vee (a+1 > 0 \wedge z \leq \text{zP})$$

We know that $(1+a) > 0$, thus we further simplify this result to:

$$(z <= \text{zP})$$

In light of this, we can generate mutants related to the variable z only. We consider the following mutants:

1. $z = (1+a) - z$
2. $z = (1+a) * z$
3. $z = (1+a)/z$
4. $z = (1+a)^z$

The above condition allows us to rule out 1,3. Thus we are left with two mutants 2 and 4.
We then choose mutant 2 and generate the new invariant relation

$$Q' = \{(s,s') | w - \frac{z}{a} == w' - \frac{z'}{a}\}$$

Using the algorithm described in [4], we find it to be compatible with the specification R. Therefore we have a program g', relatively more correct than g.Thus we don't need to process mutant 4.

7 Conclusion

To be practical, mutation analysis must be applied with a manageable number of meaningful mutants. We have illustrated how the concept of Relative Correctness can allow us to produce fewer and better mutants, leading to improved program repair. In future work, we plan on elaborating on the concept, comparing with related work and applying to large programs.

References

[1] James H. Andrews, Lionel C. Briand, and Yvan Labiche. Is mutation an appropriate tool for testing experiments? In Gruia-Catalin Roman, William G. Griswold, and Bashar Nuseibeh, editors, *27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA*, pages 402–411. ACM, 2005.

[2] Nafi Diallo, Wided Ghardallou, Ali Jaoua, Marcelo Frias, , and Ali Mili. What is a software fault, and why does it matter? [online], 2014. Available at `http://web.njit.edu/~mili/fir.pdf`.

[3] Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE Trans. Software Eng.*, 37(5):649–678, 2011.

[4] Asma Louhichi, Wided Ghardallou, Khaled Bsaïes, Lamia Labed Jilani, Olfa Mraihi, and Ali Mili. Verifying while loops with invariant relations. *IJCCBS*, 5(1/2):78–102, 2014.

[5] Ali Mili, Shir Aharon, and Chaitanya Nadkarni. Mathematics for reasoning about loop functions. *Sci. Comput. Program.*, 74(11-12):989–1020, 2009.

[6] Ali Mili, Marcelo F. Frias, and Ali Jaoua. On faults and faulty programs. In Peter Höfner, Peter Jipsen, Wolfram Kahl, and Martin Eric Müller, editors, *Relational and Algebraic Methods in Computer Science - 14th International Conference, RAMiCS 2014, Marienstatt, Germany, April 28-May 1, 2014. Proceedings*, volume 8428 of *Lecture Notes in Computer Science*, pages 191–207. Springer, 2014.

[7] Mike Papadakis and Yves Le Traon. Effective fault localization via mutation analysis: a selective mutation approach. In Yookun Cho, Sung Y. Shin, Sang-Wook Kim, Chih-Cheng Hung, and Jiman Hong, editors, *Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014*, pages 1293–1300. ACM, 2014.

Merging Termination with Abort Freedom

Wided Ghardallou[1], Nafi Diallo[2] and Ali Mili[2]

[1] Faculy of Sciences of Tunis, Tunis El Manar, Tunisia
wided.ghardallou@gmail.com
[2] CCS, NJIT, Newark New Jersey, U.S.A.
nafi.c.diallo@njit.edu,
mili@cis.njit.edu

Abstract

Termination is the property of a program to complete its execution after a finite number of operations. Abort freedom is the property of a program to complete its execution without attempting an illegal operation, such as a division by zero, an arithmetic overflow, an array reference out of bounds, a reference to a nil pointer, etc. We present an approach to the analysis of iterative programs in which these two aspects are merged into a single formula. We illustrate our approach on a number of examples, and we compare our findings to related work on termination and on abort freedom.

1 Introduction

Termination is the property of a program to complete its execution after a finite number of operations; the matter of characterizing conditions under which a loop terminates has mobilized much research attention e.g. [2]. What we refer to as abort-freedom is the property of a program to complete its execution without attempting an illegal operation (such as a division by zero, an array reference out of bounds, etc). Because the study of termination and the study of abort-freedom are interesting/challenging only in the presence of loops, we resolve to limit our attention in this paper to iterative programs. Whereas it is customary to study these two aspects as two separate properties; we believe that from a semantic standpoint, they are indistinguishable and we study them jointly (for more details, interested readers may refer to [5]). In keeping with our position, we use the term *termination* to refer to the property that the program completes its execution after a finite number of operations without causing an abort; if we want to refer specifically to the property of finite number of operations, we use the term *proper termination*. In this paper, we present an approach to analyze the termination of iterative programs. In section 2, we introduce elements of relational mathematics that we use throughout the paper, then we introduce the concept of invariant relation, and discuss how can we use this concept to analyze termination of while loops. We illustrate our approach on sample loops and compare our findings to related work. The conclusion summarizes our results and prospects.

2 Relational Definitions and Operations

In this section, we briefly present some relational notations and operations. We consider a set S defined by the values of some program variables, say x and y; we denote elements of S by s, and we note that s has the form $s =< x, y >$. We denote the x-component and (resp.) y-component of s by $x(s)$ and $y(s)$. We may use x to refer to $x(s)$ and x' to refer to $x(s')$. We refer to S as the space of the program and to $s \in S$ as a state of the program. A relation on set S is a subset of the Cartesian product $S \times S$. Constant relations on S include: the universal relation $L = S \times S$, the empty relation $\phi = \{\}$, and the identity relation denoted by I. Relations that have the form $R = C \times S$, for a subset C of S are called vectors. Operations on relations include the usual set theoretic operations of union ($R \cup R'$), intersection ($R \cap R'$), difference ($R \backslash R'$) and complement ($\overline{R} = L \backslash R$). It also includes the inverse of a

relation defined as $\widehat{R} = \{(s,s')|(s',s) \in R\}$, the product of relations R and R' denoted by RR' and defined by $RR' = \{(s,s')|\exists t : (s,t) \in R \wedge (t,s') \in R'\}$, the nth power of relation R is the relation defined by $R^0 = I$, $R^{n+1} = RR^n$, the reflexive transitive closure of relation R is defined by $R^* = \{(s,s')|\exists n \geq 0 : (s',s) \in R^n\}$, the domain of relation R is the set defined as $dom(R) = \{s|\exists s' : (s,s') \in R\}$. We leave it to the reader to check that the vector that corresponds to the domain of a relation R is nothing but RL. The range of relation R is the domain of $\widehat{R}$. The pre-restriction of relation R to predicate t is the relation defined by $\{(s,s')|t(s) \wedge (s,s') \in R\}$. As for properties of relations, we say that relation R is reflexive if and only if $I \subseteq R$, and we say that R is transitive if and only if $RR \subseteq R$. We admit without proof that the reflexive transitive closure of relation R is the smallest superset of R that is reflexive and transitive. Also, we admit that R is a vector if and only if $RL = R$. We say that R is deterministic (or that it is a function) if and only if $\widehat{R}R \subseteq I$.

3 Theoretical Background

3.1 Program Semantics

Given a program p on space S, we let P be the function of p. P is defined by the set of pairs (s,s') such that if p starts execution on s, then it terminates normally in state s'. *Normal termination* means that the program terminates after a finite number of operations, without causing an abort (due to an illegal operation) and returns a well-defined final state. Hence, the domain of P (denoted by $dom(P)$) is the set of states s such that if p starts execution on s, then it terminates *normally*. The *termination condition* of a (any) program p (including iterative programs) is the predicate $s \in dom(P)$.
We consider while loops written in some C-like programming language, and we submit the following theorem, due to [7], which we use as the semantic definition of a while loop.

Theorem 1. *Let w be a while loop of the form* `while(t)do{b}`. *Then its function W is given by:*

$$W = (T \cap B)^* \cap \overline{\overline{T}}$$

where B is the function of b and T is the vector defined by: $\{(s,s')|t(s)\}$.

The main difficulty of analyzing while loops is that we cannot, in general, compute the reflexive transitive closure of $(T \cap B)$ for arbitrary values of T and B.

3.2 Invariant Relations

If we knew how to compute reflexive transitive closures of arbitrary functions and relations, then we would apply theorem 1 to derive the function of the loop, and do away with all the methods that we use to analyze loops; but in general we do not. As a substitute, we use invariant relations, which we define as follows.

Definition 1. *Let w be a while loop of the form* `while(t)do{b}` *on space S, we say that relation R is an invariant relation for w if and only if it is a reflexive and transitive superset of* $(T \cap B)$

The interest of invariant relations is that they are approximations of $(T \cap B)^*$, the reflexive transitive closure of $(T \cap B)$; smaller invariant relations are better, because they represent tighter approximations of the reflexive transitive closure; the smallest invariant relation is $(T \cap B)^*$.

3.3 Deriving Invariant Relations

Given a while loop w of the form `while(t){b}`, we can readily compute T as the vector that defines the loop condition t, and B as the function that defines the loop body b. Using T and B, we can generate

a special invariant relation, which we call the loop's *elementary invariant relation*, according to the following formula: $R = I \cup T(T \cap B)$. The elementary invariant relation is the only invariant relation we get for free (i.e. constructively, by composing the parameters of the loop); for all other invariant relations, we have to inspect and analyze the loop in detail. To this effect, we use a compiler to map the source code of the loop (in C++) onto an internal relational notation. Because invariant relations are supersets of the function of the loop body, it is advantageous to write the function of the loop body as an intersection of term s; then, any superset of a term of the intersection is a superset of B, any superset of a pair of terms of the intersection is a superset of B, any superset of a triplet of terms of the intersection is a superset of B, etc. To automate the process of generating invariant relations, we develop a pattern matching algorithm that matches terms of the intersection against predefined patterns and, in case of success, generates invariant relations corresponding to the matched patterns. We refer to these patterns as recognizers; each recognizer is made up of variable declarations, a guard (a condition under which the match is attempted), a code template, and the corresponding invariant relation template. We classify recognizers by the number of terms of the intersection that they try to match; we refer to them as 1-recognizers, 2-recognizers or 3-recognizers, depending on whether they match one term at a time, two at a time, or three at a time; to keep combinatorics under control, we restrict ourselves to no more than three terms. The generated invariant relations are represented in the syntax of Mathematica (©Wolfram Research), to enable subsequent analysis and processing; interested readers may refer to [6] for a detailed discussion of this algorithm.

4 Theorem of Termination

Invariant relations are important for our purposes because they can be used to generate termination conditions, as provided by the following theorem, due to [6].

Theorem 2. *Let w be a while loop of the form* `while(t)do{b}` *on space S, and we let R be an invariant relation for w. Then:* $WL \subseteq R\overline{T}$

This theorem converts an invariant relation into a necessary condition of termination; if we use a small enough invariant relation, we may reach the necessary and sufficient condition of termination. Nothing in this theorem indicates whether we are modeling proper termination or abort-freedom; what determines which aspect we are modeling is the selection of the invariant relation to which we apply this theorem. Hence this theorem capture both aspects; whether the condition we derive with this theorem captures one aspect or the other or a combination of the two depends merely on what invariant relation we use. The following theorem (due to [5]) gives a general format for invariant relations that are geared towards capturing abort-freedom in while loops.

Theorem 3. *We consider a while loop w of the form* `w = while (t) {b}` *on space S, and we let* B' *be a superset of* $T \cap B$. *If* B' *satisfies the following conditions:* **(1)** B'^{+} *is anti-reflexive,* **(2)** *the following relation* $Q = \overline{B'^{*}(B'^{+} \cap V)}$ *is transitive, for an arbitrary vector V and* **(3)** $T \cap B \cap B'^{+}B' = \emptyset$.
Then $R = \overline{(B'^{*}(B'^{+} \cap \overline{BL}))}$ *is an invariant relation for w.*

As an illustration, we consider the following while loop w on integer variables i, x and y:

`while(i != 0){i=i-1; x=x+1; y=y-y/x}`

for which we derive the following invariant relations (where R_1 is the elementary invariant relation, R_2 and R_3 are generated using Recognizers form our existing database, R_4 is obtained by applying theorem 3 to the set $B' = \{(s,s')|x' = x+1\}$): $R_1 = I \cup T(T \cap B)$, $R_2 = \{(s,s')|i \geq i'\}$, $R_3 = \{(s,s')|x+i == x'+i'\}$, $R_4 = \{(s,s')|\forall h : x \leq h < x' \implies (h+1) \neq 0\}$.

By taking the intersection of these four invariant relations and applying theorem 2, we find the following necessary and sufficient condition: $(i = 0) \vee (i \geq 1 \wedge (x < -i \vee x \geq 0))$

Indeed, in order for this loop to terminate after a finite number of iterations without attempting a division by zero, either $(i = 0)$ (in which case the loop exits without iterating) or $(i > 0)$ in which case either $(x \geq 0)$ (in which case $x+1$ is initially greater than zero, and increases away from zero at each iteration) or $(x < i)$, in which case x starts negative but the loop exits before $(x+1)$ reaches 0.

5 Illustration and Comparison

We have implemented a tool that converts the source code into a relational notation, generates invariant relations by syntactic matching between the recognizers and the internal relational representation of loops, then analyzes the invariant relations in order to generate the termination condition as detailed in section 4. We ran our tool on a number of sample loops taken from various bibliographic sources that address proper termination or abort freedom conditions. In the remainder of this paper, we discuss our findings and compare them those found in the sources.

5.1 Proper Termination

We collected illustrative examples of loops from several sources (including [2]) discussing proper termination using different approaches, and have run our tool on them; we have obtained a benchmark of 36 loops, most of which operate on integer variables. When we run out tool on these examples, we find necessary conditions for 35 out of the 36 loops, and find necessary and sufficient conditions for 26 out of the 36 loops. Failure to produce sufficient conditions is usually due to the absence of relevant recognizers; this can be remedied by identifying the missing recognizers (by inspection of the relational representation of the loop) and adding them to the database. As for failure to produce (even) a necessary condition, it is merely due to the failure of Mathematica to simplify the condition generated by Theorem 2; this matter is under investigation. As a simple illustrative example, we consider the following loop, taken from [2]: `int x; while(x>=0){x=-2*x+10;}` for which our tool generates the termination condition **true**. The condition given by Terminator (tool form [2]) is $(x < 0 \vee x > 5)$ which is sufficient but unnecessary.

5.2 Abort Freedom

Because it proceeds by approximating the function of programs by increasingly smaller relations, our approach can be seen as a form of abstract interpretation [3], whereby we can make increasingly stronger claims about the functional properties of the loop as we derive smaller and smaller invariant relations. But abstract interpretation has a broader scope, since it applies to all of C, also, it is supported by tools such as Astree [1], that detects possible abort conditions in C code. It approximates the loop function with a subset (rather than a superset as in our approach), namely, the union of the first few terms (referred to as the *loop unrolling* parameter) of the transitive closure in the expression given in theorem 1. In order to show in what way what we try to achieve is distinct from Astree, we consider the following example (where a and b are arrays of N integers, x, y, i and j are integer variables):

```
while (i<N){x=x+a[i]; y=y+b[j]; j=j+i; i=i+1; j=j-i;}
```

Our tool generates the following necessary and sufficient termination condition:

$$(i \geq N) \vee (0 \leq i \leq N-1 \wedge 0 \leq j \leq N-1 \wedge 0 \leq i+j-N \leq N-1$$

We run Astree on the same example using the following initialization (which does not satisfy our termination condition): `N=50,i=0,j=599,x=1,y=2`. For a low value of loop unrolling (=3), Astree issues an alarm to the effect that array references `a[i]` and `b[j]` may be out of bounds, for a high value of loop unrolling (=700), it raises an error at the reference to `a[i]`.

LLBMC (Low Level Bounded Model Checker [4]) analyzes C and C++ programs with the aim of detecting faults in a wide class of abort conditions, such as: integer overflow, division by zero, etc. Like Astree, LLBMC requires that the code be executable, and it requires that the user specify a value of parameter unrolling before it can proceed. Also, like Astree, LLBMC alerts the user to the risk of an abort for the particular initial conditions and parameter unrolling that are selected; a different vector of initial values and a different value of unrolling may produce a different set of alerts; by contrast with both Astree and LLBMC, our approach makes a general statement about the necessary (and often sufficient) condition of termination. Like LLBMC, our approach operates on an internal representation of the program, rather than its source code; LLBMC maps source code into an internal (bit-level) representation in order to identify subtle, low-level faults that may be invisible at the code level; by contrast, the main motivation of our internal notation is to prepare the program for the generation of invariant relations. As an illustration, we consider the following loop taken from [4]: `int i,m,a[N]; while (i<m) {a[i]=i; i=i+1;}`. Our tool generates the necessary and sufficient termination condition: $(i \geq m) \vee (0 \leq i < m \leq N)$. When we run LLBMC on this example with the values `m=5,N=3,i=0` (note that these values do not satisfy our termination condition) and a loop unrolling value grater or equal to 4, it returns an error message.

6 Conclusion

In this paper, we present an invariant relation-based approach to derive termination condition of while loops. Our research partakes on the abundant work on loop termination since we define termination in the broadest sense possible to encompass not only the condition that the number of iterations is finite, but also the condition that each individual iteration completes its execution without causing an abort. We compared our approach to related work by applying it to a number of examples taken from bibliographic sources dealing with proper termination and sources dealing with abort freedom.
The tool that implements the approach discussed in this paper is under active evolution. We are currently migrating it from (inefficient) syntactic matching to semantic matching, and working on the automation of Theorem 3 in order to integrate it in the tool.

References

[1] B. Blanchet, Patrick Cousot, Radhia Cousot, J. Feret, L. Mauborgne, A. Mine, D. Monniaux, and X. Rival. The astree static analyzer. Technical report, Ecole Normale Superieure, http://www.astree.ens.fr/, 2012.

[2] B. Cook, S. Gulwani, T. Lev-Ami, A. Rybalchenko, and M. Sagiv. Proving conditional termination. In *Proceedings of the 20th international conference on Computer Aided Verification*, CAV '08, pages 328–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[3] P. Cousot and R. Cousot. An abstract interpretation framework for termination. In *Proceedings, POPL'12: 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 245–258, 2012.

[4] S. Falke, M. Merz, and C. Sinz. The bounded model chacker llbmc. In *Proceedings, 28th International Conference on Automated Software Engineering*, Palo Alto, CA, USA, 2013.

[5] W. Ghardallou, L. Labed Jilani, F. Tchier, A. Jaoua, M. Frias, J. Desharnais, and A. Mili. Computing termination conditions of while loops. Technical report, New Jersey Institute of Technology, October 2013.

[6] W. Ghardallou, O. Mraihi, A. Louhichi, L. Labed Jilani, K. Bsaies, and A. Mili. A versatile concept for the analysis of loops. *Journal of Logic and Algebraic Programming*, 81(5):606–622, May 2012.

[7] A. Mili, S. Aharon, and Ch. Nadkarni. Mathematics for reasoning about loop. *Science of Computer Programming*, pages 989–1020, 2009.

Modelling and Simulation for the Analysis of Securities Markets

Rui Hu[1,2], Vadim Mazalov[1,3] and Stephen M. Watt[1]

[1] University of Western Ontario, London, ON, Canada
{rhu8, vmazalov, Stephen.Watt}@uwo.ca
[2] Quantica Trading, Kitchener, ON, Canada
rui@quanticatrading.com
[3] Amazon Canada, Toronto, ON, Canada
mazalovv@amazon.com

Abstract

Many financial markets today are dominated by automated high-frequency trading. According to some studies, this has accounted for more than half the volume of US equity markets in recent years. High-frequency trading strategies typically adopt powerful computers and communications infrastructure and a variety of algorithms to process a large number of orders at high speed, attempting to profit sometimes a fraction of a cent on every trade. While this has led to significant theoretical and applied research, the area still presents many important challenges. These arise both in the strategy modelling phase, where accurate and efficient prediction of the price movement of securities is required, and in the evaluation phase, where strategies must be examined in a variety of market conditions before being launched in real markets. We investigate these two areas. Our modelling approach is based on clustering in a space of technical indicators, using a weighted Euclidean distance in a manner similar to certain handwriting recognition algorithms. Our evaluation environment is a market simulator that uses historical data or live data and agents to reproduce the fine-grained dynamics of financial markets. This paper outlines our approach to modelling and simulation and how they work together.

1 Introduction

Securities markets have experienced a dramatic transformation since the early 2000s. With the advancements in computer technology, traders no longer need to buy and sell securities using hand signals. Instead, their trading activities are automated by sophisticated computer algorithms, making it possible to make effective decisions to promptly react to every single market event. Powerful computers and ultra-low latency networks are also employed to accelerate data processing and message delivery, enabling them to turn over security positions very quickly in order to profit sometimes a fraction of a cent on every trade. This type of trading is commonly referred as *high-frequency trading*. Despite its increasing use and popularity [1, 3, 5, 6], high-frequency trading today still faces many critical challenges. Among these, we are particularly interested in two sub-problems pertaining to aspects of strategy modelling, where accurate and efficient prediction of securities' price movement is required, and strategy evaluation, where strategies must be examined in a variety of market conditions before being launched in real markets. In this article we present an overview of these problems and summarize our previously published solutions [4, 7].

In order to manage risk exposure and optimize profit, it is important to be able to accurately predict price movement of assets. This is a complex problem in that there are a considerable number of factors that may affect price in securities markets. Moreover, these factors themselves may have complicated cause-and-effect relationships, resulting in a variety of potential outcomes. We are interested in the problem of providing predictions of the price movement in the short term, as opposed to the long term, since the number of contributing factors is smaller, and their values are easier to measure. Also, the

possibility of a impact by outside factors, e.g. an unscheduled news release, is lower and has less effect in the short-term. In particular, we present a trading model that can provide accurate and efficient short-term prediction of *one* change in the price of an asset. Our method monitors the market and sets up a space of observations, then measures how close the real-time data is to the recorded observations. Predictions are made based on numerical indicators computed from data received from the market. This work has been reported in [7].

At the same time, trading strategies must be evaluated for correctness and performance before using them real market. This in practice is carried out through simulators which significantly rely on real-time market data or historical data. While this can provide traders with valuable information, there are a number of pitfalls. First, live market data is not always available, which restricts the use of simulators to certain market hours. In addition, the trading strategies tested do not have any impact to the market as they can only follow the trend and their orders are simply executed based on the current market conditions. Similar issues also exist in back-testing approaches. Last, but not least, existing simulators typically do not provide a standard protocol for interaction with users. Instead, they require skills in specific programming languages and demand trading strategies to be implemented on top of proprietary Application Program Interfaces (APIs). This can restrict the evaluation of trading strategies to a single simulation environment. To address this problem, we present a simulator that can support market simulation research and is suitable for strategy evaluation. The simulator is independent of any particular data feed and can provide a realistic testing environment by reproducing certain phenomena of a real market. Multiple users can connect to the simulation server at the same time, allowing them to not only assess the viability of their trading strategies using pre-defined market conditions, but also to create very specific ones that suit their needs. This work has been reported in [4].

The remainder of the article is organized as follows. In Section 2 we describe an example of prediction model for high-frequency trading. Section 3 presents a simulator that is suitable for evaluation of trading strategies. In Section 4 we conclude the article with a brief discussion.

2 A Trading Model

Technical Indicators The number of possible technical indicators that can be extracted from the stock market is potentially overwhelming so careful manual or algorithmic selection of indicators is important. Indicators should not be redundant and should sufficiently describe the asset at the time of an event. In order to limit the available indicators to a manageable number and predict a price change, we examine only the quotes at the current best bid and ask prices. We consider indicators that describe the activity of a single product independently of complementary and supplementary securities. We compute the following indicators for each exchange:

- Weighted rate of change (*ROC*) of the relation of the bid depth (number of shares bid) to the offer depth (number of shares offered).
- n_{cb} (n_{co}) – the number of times an exchange locked the market on the bid (offer).
- n_{lb} (n_{lo}) – the number of times an exchange left the National Best Bid and Offer (NBBO) on the bid (offer).

Within each exchange, each of the indicators is assigned a weight. The weight of an exchange is determined as the average weight of its indicators. We then compute the composite indicators, s_b (s_o), which are the sum of weights of exchanges whose bid (offer) price is equal to the NBBO.

To remove outliers we use the three-sigma rule, assuming that the values of indicators are normally distributed. After the removal of outliers, the values of indicators are normalized by a transformation on the feature values to map them to the range $[0,1]$: we find the maximum x_i^M and the minimum x_i^m values of an indicator i among all points in the cluster, and then normalize the feature as

$$x_i' = \frac{x_i - x_i^m}{x_i^M - x_i^m}.$$

Algorithm 1 ComputeWeights(X)

Input: X – a cluster of normalized points.
Output: w – a vector of weights of features.

$s \leftarrow \sum_{i=1}^{D} \sigma_i$
{where σ_i is the standard deviation of feature i in the cluster X}
for $i = 1$ to D **do**
  $w_i \leftarrow 1 - \sigma_i / s$
end for
return w

Algorithm 2 Predict(x)

Input: x, a normalized D-dimensional feature point to be classified.
  d, a distance threshold. r, distance ratio threshold
Output: Either "predict price change up" or "predict price change down" or "not classified".

{Compute the squared distances to the centroids c^d and c^u of clusters down and up respectively, where w_i^d and w_i^u are the weights of the i-th indicator in the corresponding clusters.}
$d_d \leftarrow \sum_{i=1}^{D} w_i^d (x_i - c_i^d)^2$; $d_u \leftarrow \sum_{i=1}^{D} w_i^u (x_i - c_i^u)^2$

if $d_d < d$ **and** $d_d / d_u < r$ **then**
  return "predict a price change down"
end if
if $d_u < d$ **and** $d_u / d_d < r$ **then**
  return "predict a price change up"
end if
return "not classified"

For a point p to be an outlier, at least one of its coordinates p_o should satisfy $|p_o - s_o| > 3\sigma_o$. The case $p_o - s_o > 3\sigma_o$ implies

$$p_o > \frac{1}{K}\sum_{i=1}^{K} x_{oi} + 3\sqrt{\frac{1}{K}\sum_{i=1}^{K}(x_{oi} - \frac{1}{K}\sum_{j=1}^{K} x_{oj})^2}, \tag{1}$$

where K is the number of points in a cluster and x_{ij} is the i-th feature of the j-th point. Applying inequality (1) to normalized features, we have

$$\frac{p_o - x_o^m}{x_o^d} > \frac{1}{K}\sum_{i=1}^{K} \frac{x_{oi} - x_o^m}{x_o^d} + 3\sigma_o', \tag{2}$$

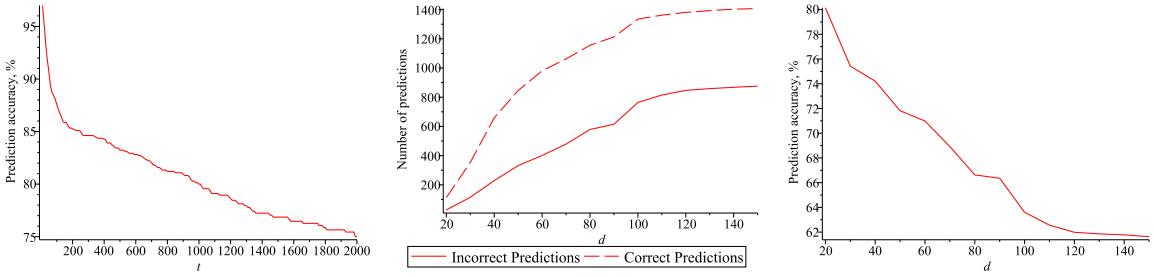
where

$$\sigma_o' = \sqrt{\frac{1}{K}\sum_{i=1}^{K}\left(\frac{x_{oi} - x_o^m}{x_o^d} - \frac{1}{K}\sum_{j=1}^{K}\frac{x_{oj} - x_o^m}{x_o^d}\right)^2}.$$

An analogous reasoning can be applied to the inequality $p_o - s_o < -3\sigma_o$.

Computation of Weights Each indicator within a cluster is assigned a weight, computed as shown in Algorithm 1. The weight of an exchange is computed as the average of the weights of its indicators.

The Classifier To analyze incoming quotes at high frequency in real-time, the recognition model should be computationally efficient. For the classifier, we have chosen to compute the feature-weighted distance



(a) (b) (c)

Figure 1: Prediction results:
(a) Prediction accuracy as a function of t (in milliseconds),
(b) The number of correct and incorrect predictions depending on the distance threshold d, and
(c) Prediction accuracy as a function of the distance threshold d

from a test point to the centroid of a cluster, since this is one of the least expensive techniques in artificial intelligence.

This technique is fast in training and classification. To *train* the model, one needs to collect a certain number of points in clusters and find the centroid of each cluster. We are interested in two classes of points: one class for price changes up and another class for price changes down. *Classification* of a quote is performed by computing the squared weighted Euclidean distance from the feature-vector of the quote to the centroid of each training cluster, as shown in Algorithm 2.

As can be observed in the algorithm, we differentiate the notions of classification and prediction. Classification happens with each quote received – a feature vector is formed and the distances to centroids are evaluated. In contrast, a *prediction* is made only if the distances between the sample and the centroids satisfy certain criteria, i.e. if the feature point is relatively close to one of the two centroids. The prediction accuracy can be increased and the number of predictions decreased by reducing the thresholds d and r in Algorithm 2. The necessary values of the thresholds can be determined empirically.

Experimental Setting The experiments were performed on data for MSFT (Microsoft) securities, using quotes from the following exchanges: NYSE Archipelago Exchange (Arca), Better Alternative Trading System (BATS) BZX Exchange, BATS BYX Exchange, Chicago Board Options Exchange (CBOE), Direct Edge A (EDGA), Direct Edge X (EDGX), NASDAQ, NASDAQ OMX BX, National Stock Exchange, and NASDAQ PHLX. The recorded events include: change in bid/offer prices and bid/offer depth. We recorded several days in December, 2011 with the total of 9,389,993 quotes and 4,658 price changes. Training was performed until both clusters had at least 10 points. The value of the weight in computation of the *ROC* was taken as 0.6. After 5 changes in price, parameters of a cluster were recomputed.

We calculated two measures: *on-change* distance accuracy and *prediction* accuracy. The on-change distance measure was counted as correct if the distance to the centroid of the cluster in the direction of the price change was smaller than the distance to the other cluster. In other words, if a change down (up) was recorded and $d_d < d_u$ ($d_d > d_u$), the on-change distance measure was counted as correct.

The prediction accuracy with a prediction interval of t was computed as follows. If the prediction was in the direction of the price change, and the interval between a prediction and the actual change was greater than t, the count of correct predictions was increased by one. If the interval was less than t, the count was not changed. If the change happened in the opposite direction, the count of wrong predictions

was increased by one, independently of the time interval after the prediction. This measure aimed to simulate real-life trading, when execution of a transaction takes a certain amount of time, depending on infrastructure.

Experimental Results The on-change accuracy of the model on the recorded data was 96.25%. The prediction accuracy as a function of t is presented in Figure 1(a) for the distance thresholds $d = 20$ and $r = 1/20$. There are 28 incorrect predictions for any t. We also measured the number of correct and incorrect predictions depending on the distance threshold d with fixed $r = 1/20$ for $t = 1000ms$. The results are presented in Figure 1(b). Figure 1(c) shows the prediction accuracy as the function of d.

3 A Market Simulator

We now present a simulator that can support market simulation research and is suitable for evaluation of algorithmic trading strategies.

Simulator Design The simulator currently consists of a matching engine, a communication interface and a variety of simulated trading agents. The matching engine accepts orders from both logged in users and computerized agents. It maintains a number of order books, each of which records the interest of buyers and sellers in a particular security and prioritizes their orders based on their price and arrival time. This centralized order system continuously attempts to match buy and sell orders. Matching rules are implemented based on continuous double auctions. The matching engine also publishes quote updates to all subscribers, which is handled by the communication interface.

To allow multiple users to interact with our simulator simultaneously and independently, we use the Financial Information eXchange (FIX) protocol [2], which is the *de facto* communications standard in global financial markets. The simulator provides each user a designated port for login and maintains a dedicated channel for communication. Both inbound and outbound messages, such as orders and execution reports, are encoded as FIX format. Using the FIX protocol also provides easy access to our simulator. Most users in both industrial and academic algorithmic trading settings are familiar with the FIX protocol or have it already implemented in order to connect to financial markets. This makes it possible to interact with our simulator with little modification to their systems.

In order to create various market conditions that are suitable for testing, we have developed five pre-defined types of simulated trading agents that represent an important subset of trading entities we observe in the real market. These agents are able to adapt to the market and interact with users' algorithmic trading strategies. The first of these is Market Maker Agent which plays neutrally against the market. Its primary objective is to enhance the liquidity and the depth of the market, resulting in a stable market. In contrast, Liquidity Taker Agent takes liquidity from the market by posting market orders that are often immediately executed at the best available price. By increasing the size or the frequency of the orders, it can potentially cause the quoted prices to change dramatically. Figure 2 shows a comparison of volatility between two simulations. The settings were the same except in the second simulation the Liquidity Taker Agent issues market orders at a higher frequency. Similar to the Market Maker Agent, Liquidity Provider Agent places limit orders to the market. By posting limit orders on both sides without immediately triggering a trade, it adds liquidity to the order book and consequently increases the depth and the stability of the market. We have also developed a Random Agent which uses no information about the market and issues random orders at certain time intervals. This type of agent can be used to create chaos in the simulation environment as well as to investigate the cause of certain market phenomena. The last type of agent is Swift Agent. Compared to the other four agents, a Swift Agent is more sophisticated in that it is able to control the number of open orders it places. This prevents the agent from exposing itself to too much risk, just as human traders would also do in a real market. In addition, the agent is able to monitor the price fluctuations of the simulated market. If the price variation exceeds a certain threshold, the agent will attempt to place more orders on the opposite side to counter the trend.

Software Implementation The simulator is intended to be useful to evaluate trading strategies. It sup-

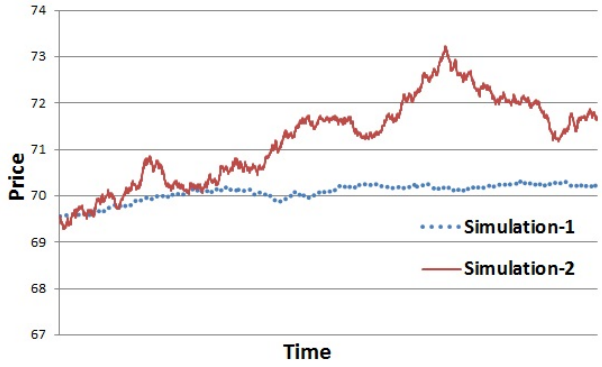


Figure 2: A comparison of two simulations with different liquidity taking.

ports a variety of security types, including equities, futures, foreign exchange, and options. The simulator may run a market consisting exclusively of simulated (human or robotic) trading agents using different trading strategies (e.g., to study algorithms or market effects), or external participants (typically human) may log in and interact with the simulated market (e.g., for training). Participants (logged in users or simulated trading agents) can submit both limit and market orders with different time-in-force, allowing them to interact with the simulation environment as if they were trading in a real market. At the same time, the simulator adopts the FIX protocol, which allows multiple users to interact with the simulation environment simultaneously and independently. In contrast to other simulators that require testing trading strategies to be built on top of proprietary APIs, our simulator uses open protocols so can be integrated easily into their systems with little modification. The simulator is able to run in two different settings, each of which is useful in certain scenarios.

The first setting uses simulated trading agents, each of which is able to adapt and react to real-time market events by following selected pre-defined strategies. All of the agents are configurable. By adjusting their configurations, we can create very specific market conditions that would occur only rarely in a real market. In addition, the agent-based simulator is able to run at any time as the data are generated by the computerized agents. The agent-based simulator is useful in that it allows users to create desired market conditions where they can test their trading strategies whenever it is needed.

In the second setting, the simulator receives live market data from real exchanges and broadcasts this data to each user. Orders that are submitted by users are executed based on the current market conditions. This type of simulator is claimed by some to be more realistic. Meanwhile, the real market data simulator, in practice, has access to all the products available in the markets, while in agent mode this is not feasible unless there is a further configuration. We provide both these settings to users to evaluate their trading strategies, and give them the freedom to choose the one that is most suitable for their needs.

Useful Scenarios Both the agent-based and the live-data simulator have been adopted by Quantica Trading, a company located in Kitchener, Canada, which develops algorithmic trading software. We have found, informally, the agent-based and live-data simulators to be useful in a number of scenarios. First, some conditions, such as a market crash, may not occur very often in a real market, but they are extraordinarily costly when they do occur and so must be examined. With the agent-based simulator, we can easily reproduce these conditions, and variants, to test strategies. In addition, the simulator has also been found useful for education and training purposes. By executing trades in the risk-free simulation environment, it facilitates trading drills designed for new traders, allowing them to learn how to execute trades and manage risk faster. Moreover, the simulator is also suitable for software demonstration. With round-the-clock access and risk-free testing, users can present demonstrations of their software

applications at their convenience. Last, but not least, a simulator of the type we present also benefits users beyond the high-frequency trading world. High-quality simulation is essential to improve the regulatory environment for North American markets. At the moment, the true impact of regulation cannot be completely understood until it is in effect in the markets. This means that regulation can have unintended consequences or not achieve its desired results. High-quality simulation can help improve this, reducing risk of events such as the "Flash Crash" of 2010 [8].

4 Conclusion

We have explored both the analysis and simulation of financial markets. Our analysis has explored short-term price changes of securities using pattern recognition techniques. This method is based on the distance to the centroid of a set of feature vectors, representing an empirical combination of indicators. The model demonstrates good performance, even with naïve indicators. We have also presented a financial market simulator that supports a full range of security types and allows users to interact as if they were trading in a real market. It uses FIX as the communication protocol, allowing multiple users to interact with the simulation environment simultaneously and independently. We have also presented several types of simulated trading agents which represent a subset of traders observed in real markets. All of these agents are configurable and, by adjusting their parameters, very specific market conditions can be created to explore certain market behaviours. We have found that in a corporate setting that our simulator is useful in a number of scenarios, including system testing, education, training, and policy evaluation.

Acknowledgments

This work was supported, in part, by a CU-I2I grant from the Natural Sciences and Engineering Research Council of Canada. We thank James McInnes, Jonathan Leaver Travis Felker, and Peter Metford for discussions relating to desired function and implementation.

References

[1] Rob Curran and Geoffrey Rogow. Rise of the (Market) Machines. http://blogs.wsj.com/marketbeat/2009/06/19/rise-of-the-market-machines/. [retrieved: Oct 2014].

[2] FIX Trading Community. Financial Information eXchange (FIX) Protocol. http://www.fixtradingcommunity.org/. [retrieved: Oct 2014].

[3] Terry Hendershott, Charles M. Jones, and Albert J. Menkveld. Does Algorithmic Trading Improve Liquidity? *J. Finance*, 66(1):1–33, Feb 2011.

[4] Rui Hu and Stephen M. Watt. An Agent-Based Financial Market Simulator for Evaluation of Algorithmic Trading Strategies. *6th International Conference on Advances in System Simulation*, pages 221–227, Oct 2014.

[5] Rob Iati. The Real Story of Trading Software Espionage. http://www.wallstreetandtech.com/trading-technology/the-real-story-of-trading-software-espionage/a/d-id/1262125? [retrieved: Oct 2014].

[6] Bank of England. Patience and Finance. http://www.bis.org/review/r100909e.pdf. [retrieved: Oct 2014].

[7] Vadim Mazalov Travis Felker and Stephen M. Watt. Distance-Based High-Frequency Trading. *14th International Conference on Computational Science*, 29:2055–2064, July 2014.

[8] US Commodity Futures Trading Commission and US Securities & Exchange Commission. Findings Regarding the Market Events of May 6, 2010. http://www.sec.gov/news/studies/2010/marketevents-report.pdf. [retrieved: Oct 2014].

Symbolic Algorithm for Construction of Toric Compactifications

Alexey A. Kytmanov and Alexey V. Shchuplev

Siberian Federal University, Krasnoyarsk, Russia
aakytm@gmail.com, alexey.shchuplev@gmail.com

Abstract

Given a toric variety we present an algorithm that constructs a "larger" toric variety such that the initial one can be embedded in it as an infinite part. This compactification of an affine space generalizes the well-known decomposition $\mathbb{P}_n = \mathbb{C}^n \sqcup \mathbb{P}^{\infty}_{n-1}$.

1 The Theoretical Result

The well-known representation of the projective space $\mathbb{P}_{n+1} = \mathbb{C}^{n+1} \sqcup \mathbb{P}_n$ as an affine space $\mathbb{C}^{n+1}$ with $\mathbb{P}_n$ attached "at the infinity" admits the following interpretation. The closure $\bar{l}$ in $\mathbb{P}_{n+1}$ of every line l in $\mathbb{C}^{n+1}$ intersects the attached $\mathbb{P}_n = \mathbb{C}^{n+1} \setminus \{0\}/\sim$ at the point $[l]$ repesented by l. This relates the projective space with the notion of linear perspective.

This interpretation turns out to be a special case of a general phenomenon of toric varieties. Indeed, any smooth simplicial toric variety can be embedded in a larger toric variety in the manner that is called a multidimensional perspective.

More precisely, assume that a complete fan Σ in $\mathbb{R}^n$ contains at least one simple n-dimensional cone, then the following theorem proved in [3] holds.

Theorem 1. *Let* Σ *be a simplicial complete fan in* $\mathbb{R}^n$ *with* d *generators. There exists a* d*-dimensional simplicial and compact toric variety*

$$X_{\widetilde{\Sigma}} = \mathbb{C}^d \sqcup (\mathscr{X}_1 \cup \cdots \cup \mathscr{X}_r)$$

with 'infinite' toric hypersurfaces $\mathscr{X}_1, \ldots, \mathscr{X}_r$ *such that its 'skeleton'* $\mathscr{X}_1 \cap \cdots \cap \mathscr{X}_r$ *is isomorphic to* X_Σ*. Moreover, for every* $\zeta \in \mathbb{C}^d \setminus Z(\Sigma) \subset X_{\widetilde{\Sigma}}$ *the closure* $\overline{G \cdot \zeta}$ *of its orbit in* $X_{\widetilde{\Sigma}}$ *intersects 'the skeleton of infinity' in a unique point corresponding to the class of* ζ *under the isomorphism.*

This geometric construction proves useful when constructing new integral representations and residues ([2], [1]).

2 The Algorithm

We now give a description of an algorithm that constructs a "larger" toric variety such that the given a toric variety can be embedded in it as an infinite part.

Algorithm BigVariety (*vec_list*, *cone_list*)
Input: List of vectors — 1-dimensional generators of initial "small" toric variety *vec_list*, list of numbers of vectors that generate cones of maximal dimension of initial "small" toric variety *cone_list*.
Output: List of vectors — 1-dimensional generators of the corresponding "big" toric variety , list of numbers of vectors that generate cones of the maximal dimension of the corresponding "big" toric variety.

Step 1. Construction of 1-dimensional generators.
for i **from** 1 **to** n **do**
e_i := add to vec_list_i $(d-n)$ zeros
end do
for i **from** $n+1$ **to** d **do**
e_i := empty vector
e_i := add to e_i $(i-1)$ zeros
e_i := add to e_i 1
e_i := add to e_i $(d-i)$ zeros
end do
for i **from** $n+1$ **to** d **do**
v_i := add to vec_list_i $(d-n)$ zeros
v_i := v_i - e_i
end do

Step 2. Construction of cones of the maximal dimension.
$I := \{1,\dots,n\}$
$J := \{n+1,\dots,d\}$
for i **from** 1 **to** number of elements in *cone_list* **do**
K := intersection of $cone\_list_i$ and I
L := intersection of $cone\_list_i$ and J
QS := **Split** $(J \setminus L)$
for j **from** 1 **to** number of elements in QS **do**
$KLQS$:= add to $KLQS$ list of elements $[K, L, QS_j]$
end do
end do
for i **from** 1 **to** number of elements in $KLQS$ **do**
EV := add to EV list of
elements $[(KLQS_i)_1 \cup (KLQS_i)_2 \cup (KLQS_i)_3, (KLQS_i)_2 \cup (KLQS_i)_4]$
end do
CL := empty list
for i **from** 1 **to** number of elements in EV **do**
cone := empty list
for j **from** 1 **to** number of elements in $(EV_i)_2$ **do**
cone := add to *cone* element $((EV_i)_2)_j + d - n$
end do
CL := add to CL $[(EV_i)_1, cone]$
end do

return([list of e_i, list of v_i], CL)

The procedure **Split** (S) gives all possible combinations of sets S_1, S_2 such that $S_1 \cup S_2 = S$.

Split := **procedure** (S)
P := set of all subsets of S
for i **from** 1 **to** number of elements in P **do**
t_i := list of $[S_i, S \setminus S_i]$
end do
return(list of t_i)
end procedure

3 Examples

The following examples were used to test the above-described algorithm of construction the "larger" toric variety for each given "smaller" variety. The algorithm was implemented in Maple computer algebra system.

Example 1. *For $\mathbb{P}_2$ the "larger" toric variety is $\mathbb{P}_3$. In general, for $\mathbb{P}_n$ which fan is given by the vectors*

$$v_1 = (\underbrace{1,0,\ldots,0}_{n}),\ldots,v_n = (0,\ldots,0,1),v_{n+1} = (\underbrace{-1,\ldots,-1}_{n})$$

where each three vectors generate a cone od maximal dimension, the "larger" toric variety will be $\mathbb{P}_{n+1}$ with the fan generated by

$$e_1 = (\underbrace{1,0,\ldots,0}_{n+1}),\ldots,e_{n+1} = (0,\ldots,0,1),v_{n+1} = (\underbrace{-1,\ldots,-1}_{n+1})$$

where each three vectors generate a cone od maximal dimension as well.

Example 2. *For $\mathbb{P}_1 \times \mathbb{P}_1$ with the fan given by*

$$v_1 = (1,0),v_2 = (0,1),v_3 = (-1,0),v_4 = (0,-1)$$

with 2-dimensional cones generated by

$$\langle v_1,v_2\rangle,\langle v_2,v_3\rangle,\langle v_3,v_4\rangle,\langle v_4,v_1\rangle,$$

the "larger" toric variety is $\mathbb{P}_2 \times \mathbb{P}_2$. Its fan is spawned by the 4-dimensional vectors

$$e_1 = (1,0,0,0),e_2 = (0,1,0,0),e_3 = (0,0,1,0),e_4 = (0,0,0,1),$$
$$v_3 = (-1,0,-1,0),v_4 = (0,-1,0,-1)$$

where 4-dimensional cones are generated by

$$\langle e_1,e_2,v_3,v_4\rangle,\langle e_1,e_2,e_3,v_4\rangle,\langle e_1,e_2,e_4,v_3\rangle,\langle e_1,e_2,e_3,e_4\rangle,\langle e_2,e_3,v_3,v_4\rangle,$$
$$\langle e_2,e_3,e_4,v_3\rangle,\langle e_3,e_4,v_3,v_4\rangle,\langle e_1,e_4,v_3,v_4\rangle,\langle e_1,e_3,e_4,v_4\rangle.$$

In general, for $(\mathbb{P}_1)^n$ the "larger" toric variety will be $(\mathbb{P}_2)^n$.

Example 3. *For the so called hybrid of $\mathbb{P}_2$ and $\mathbb{P}_1 \times \mathbb{P}_1$ — a toric variety given by the fan*

$$v_1 = (1,0),v_2 = (0,1),v_3 = (-1,0),v_4 = (0,-1),v_5 = (-1,-1)$$

with 2-dimensional cones generated by

$$\langle v_1,v_2\rangle,\langle v_2,v_3\rangle,\langle v_3,v_4\rangle,\langle v_4,v_5\rangle,\langle v_5,v_1\rangle,$$

the resulting variety will be given by the fan

$$e_1 = (1,0,0,0,0),e_2 = (0,1,0,0,0),e_3 = (0,0,1,0,0),e_4 = (0,0,0,1,0),$$
$$e_5 = (0,0,0,0,1),v_3 = (-1,0,-1,0,0),v_4 = (0,-1,0,-1,0),v_5 = (-1,-1,0,0,-1)$$

with 5-dimensional cones generated by the following 5-tuples

$$\langle e_1,e_2,v_3,v_4,v_5\rangle,\langle e_1,e_2,e_3,v_4,v_5\rangle,\langle e_1,e_2,e_4,v_3,v_5\rangle,\langle e_1,e_2,e_5,v_3,v_4\rangle,\langle e_1,e_2,e_3,e_4,v_5\rangle,$$
$$\langle e_1,e_2,e_3,e_5,v_4\rangle,\langle e_1,e_2,e_4,e_5,v_3\rangle,\langle e_1,e_2,e_3,e_4,e_5\rangle,\langle e_2,e_3,v_3,v_4,v_5\rangle,\langle e_2,e_3,e_4,v_3,v_5\rangle,$$
$$\langle e_2,e_3,e_5,v_3,v_4\rangle,\langle e_2,e_3,e_4,e_5,v_3\rangle,\langle e_3,e_4,v_3,v_4,v_5\rangle,\langle e_3,e_4,e_5,v_3,v_4\rangle,\langle e_4,e_5,v_3,v_4,v_5\rangle,$$
$$\langle e_3,e_4,e_5,v_4,v_5\rangle,\langle e_1,e_5,v_3,v_4,v_5\rangle,\langle e_1,e_3,e_5,v_4,v_5\rangle,\langle e_1,e_4,e_5,v_3,v_5\rangle,\langle e_1,e_3,e_4,e_5,v_5\rangle.$$

Acknowledgments

This work was partially supported by the state order of the Ministry of Education and Science of the Russian Federation for Siberian Federal University, task 1.1462.2014/K (first author).

References

[1] A. A. Kytmanov and A. Y. Semusheva. Averaging of the cauchy kernels and integral realization of the local residue. *Mathematische Zeitschrift*, 264(1):87–98, 2010.

[2] A. Shchuplev. On reproducing kernels in $\mathbb{C}^d$ and volume forms on toric varieties. *Russian Mathematical Surveys*, 60(2):373–375, 2005.

[3] A. Shchuplev, A. Tsikh, and A. Yger. Residual kernels with singularities on coordinate planes. *Proceedings of Steklov Institute of Math.*, 253(2):256–274, 2006.

Automated Detection and Resolution of Firewall Misconfigurations

Amina Saâdaoui, Nihel Ben Youssef Ben Souayeh and Adel Bouhoula

Digital Security Research Unit
Higher School of Communication of Tunis (Sup'Com)
University of Carthage, Tunisia
{amina.saadaoui, nihel.benyoussef, adel.bouhoula}@supcom.tn

Abstract

Firewalls provide efficient security services if they are correctly configured. Unfortunately, configuring a firewall is well known highly error prone and work-intensive if performed manually and become infeasible in the presence of a large number of rules in a firewall configuration. Therefore, there is a need of automated methods to analyze, detect and correct misconfigurations. Prior solutions have been proposed but we note their drawbacks are threefold: First, common approaches deal only with pairwise filtering rules. In such a way, some other classes of configuration anomalies could be uncharted. Second, they did not distinguish the intended firewall conflicts from the effective misconfigurations. Third, although anomalies resolution is a tedious task, it is generally given to the network administrator.
We present, in this paper, a formal approach whose contributions are the following: Detecting new classes of anomalies, bringing out real misconfigurations and finally, proposing automatic resolution method by considering the security policy. We prove the soundness of our method. The first results we obtained are very promising.

1 Introduction

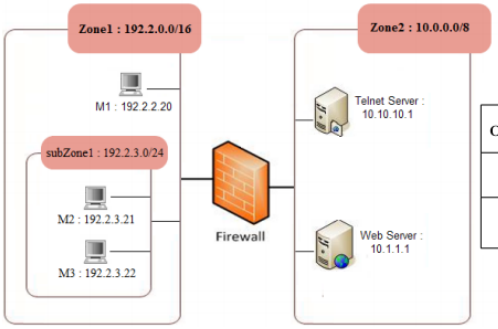
The function of the firewall is to supervise data flow and to decide what to accept or to reject based on an ordered list of filtering rules defined by the network administrator according to the requirement of the global security policy. Unfortunately, configuring firewall rules is a tedious task due to, essentially, the complexity and interdependency of filtering rules. As an example, consider a typical enterprise network shown in Fig. 1. The global security policy that should be implemented is described as follows:

- Machine M1 cannot access Zone2.
- All users in Zone1 except M1 can access the Web Server.

We can note that the rules of the firewall configuration are consistent with the global security policy *SP* as filtering rules of firewalls are, generally, processed from the top down, and the first match wins. Although no misconfigurations are identified, most related studies [6, 16, 5] present the conflict between rules r_1 and r_2 as a purely syntactic anomaly, called *Correlation*, since these two rules handle common packets with different actions.

Once reporting a conflict, the decision is left up to the network administrator to decide whether it is a misconfiguration. If it is the case, he should correct the configuration by ensuring not altering the firewall behavior and not violating the security policy. This task is more complex than it appears at first glance especially when a large number of filtering rules are deployed.

There are numerous studies [6, 5, 11, 10, 16, 12] on anomalies detection and resolution in firewall configuration. In these methods, detection algorithms presented are based on the analysis of pairwise rules. In this way, errors due to the union of rules are not explicitly considered. Hu et al [8, 9] propose



Firewall Configuration

Order	Srce_@	Dest_@	Protocol	Dest_Port	Action
r1	192.2.2.20	10.0.0.0/8	*	80	deny
r2	192.2.0.0/16	10.1.1.1	TCP	80	accept

Figure 1: Network topology

a new anomaly management framework (FAME). The proposed idea to resolve anomalies is based on calculating a risk level that permits, in some cases, users to manually select the appropriate strategies for resolving the conflict. In such a way, the administrator can make wrong choices. Firewall Builder [7] and Athena Firepac [3] provide a detection of anomalies in the set of filtering rules. Nevertheless, this discovery mechanism only detects trivial equality or inclusion or shadowing between the filtering rules and did not provide an automatic resolution of the detected conflicts. In [1] authors present a firewall analysis engine named Fang, based on a combination of a graph algorithm and a rule-base simulator.

Some interesting work has been done on firewall configuration and security policy verification [2, 15, 13]. Alex X. Liu [2] proposes a firewall verification method. But, all the work on this paper is on a conflict free configuration. Or, in most cases firewall rules are conflicted. Matsumoto and Bouhoula [15] propose a SAT based approach for verifying firewall configurations with respect to the security policy requirements. Ben Youssef and Bouhoula [13] propose an automatic method for checking whether a firewall is well configured according to a global security policy. This formal method does not allow to automatically correct anomalies. FINSAT [14], [4] incorporates ACL (Access Control List) conflict analysis procedure for detecting various types of ACL rule conflicts in the model using Boolean satisfiability (SAT) analysis. The conflicts are reported as ”error(s)” in case of SAT result with satisfiable instances. Then, the Network administrator need to reconfigure by himself the ACL rules depending on the results because the tool does not provide an automatic correction.

In this paper, we propose new techniques allowing the automatic detection and correction of centralized-firewall misconfigurations. Our approach allow to detect new classes of anomalies and help the network administrator to automatically distinguish and correct effective misconfigurations.

This paper is organized as follows: Section 2 overviews the typical definitions of anomalies and the formal representation of firewall configurations and security policies. In section 3, we present our new classification of filtering rules anomalies. In section 4, we articulate our approach to detect and resolve firewall misconfigurations. Finally, we present our conclusions and discuss our plans for future work.

2 Formal Definitions

2.1 Firewall Configuration

We consider a finite domain $\mathscr{P}$ containing all the headers of packets possibly incoming to or outgoing from a network.
A simple firewall configuration is a finite sequence of filtering rules of the form $FR = (r_i \Rightarrow A_i)_{0<i<N+1}$. These rules are tried in order, up to the first matching one. A filtering rule consists of a precondition

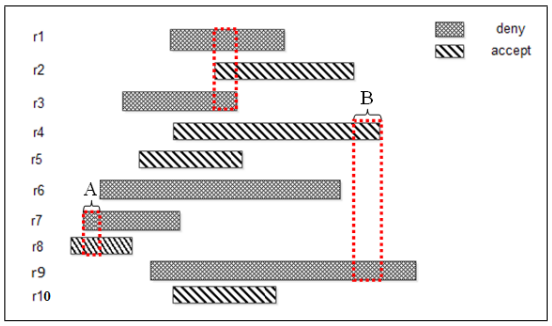


Figure 2: Firewall configuration

r_i which is a region of the packet's space, usually, consisting of source address, destination address, protocol and destination port. Each right member A_i of a rule of FR is an action defining the behavior of the firewall on filtered packets: $A_i \in \{accept, deny\}$.
We consider the function $dom(r_i)$ which maps each r_i into the subset of $\mathscr{P}$ and represents the domain of r_i (Set of packets handled by r_i). Fig. 2 illustrates a schematic Firewall configuration where domain of each filtering rule is depicted as rectangles and actions by different fillings.

2.2 Security Policy

A security policy (SP) is a set *SP* of formulas defining whether packets are accepted or denied. We consider a security policy which is presented as a finite unordered set of directives. For example, a directive could be as follows:

- A network net1, except the machine A, has the right to access to the FTP service provided by a server S located in the network net2.

Definition (*Absence of misconfigurations) A firewall configuration is compatible with SP if and only if the action handled by FR is the same as defined in SP.*

3 Our Classification of Anomalies

We divide anomalies in two principal classes: Class1 (class based on superfluous rules) and Class2 (class based on conflicting rules).

3.1 Superfluous Rules-Class Anomalies

A rule is superfluous if and only if it is redundant to other subsequent rule(s), shadowed by previous rule(s) or partially shadowed/Partially redundant.

3.1.1 Shadowing

A rule is shadowed when some/all *previous* rules match all the packets that match this rule, such that the shadowed rule will never be selected. In other words, r_i is shadowed by a finite sequence of rules T($T = \cup_{j<i}\{rj\}$) iff the domain of r_i is included in the domain of T. Formally:
A rule r_i is shadowed iff for all packets $p \in \mathscr{P}$, if $p \in dom(r_i)$ then there exists at least a rule r_j with

$j < i$ *such that* $p \in dom(r_j)$.
For example, in Fig. 2, we can note that r_5 is shadowed by its preceding rules.

3.1.2 Redundancy

A rule r_i is redundant to some/all *subsequent* rules if the action undertaken by this rule r_i for all packets it matches, will be the same once removed. Formally:
A rule $\{r_i \Rightarrow A\}$ *is redundant iff for all packets* $p \in \mathscr{P}$, *if* $p \in dom(r_i) \setminus \cup_{j<i}\{dom(r_j)\}$. *Then, there exists at least one rule* $\{r_k \Rightarrow A\}$, *such that* $p \in dom(r_k) \setminus \cup_{i<m<k}\{dom(r_m)\}$.
This is the case for rule r_6 which is redundant to r_7 and r_9 in the firewall configuration of Fig. 2. All packets in $dom(r_6)$ will be accepted by r_7 and r_9.

3.1.3 Partially Shadowing/Partially Redundancy

A rule r_i is partially shadowed /partially redundant when some/all previous rules match some packets that match this rule, given that the shadowed part of the rule will never be selected and the action undertaken by this rule r_i for the other packets (not matched by shadowed part) will be the same once removed. Formally:
A rule r_i *is partially shadowed /partially redundant iff for all packets* $p \in \mathscr{P}$, *if* $p \in dom(r_i)$ *then there exists at least a rule* r_j *with* $j < i$ *such that* $p \in dom(r_j)$. *Or, there exists at least one rule* $\{r_k \Rightarrow A\}$, *such that* $p \in dom(r_k) \setminus \cup_{i<m<k}\{dom(r_m)\}$.
For example, in Fig. 2, we can note that r_2 is partially shadowed by r_1 and partially redundant to r_4.

3.2 Conflicting Rules-Class Anomalies

Existing conflict classification methods [6, 11] only consider a conflict between two rules as an inconsistent relation between these rules without considering the entire firewall configuration. For example, in Fig. 2, the common existing classifications report that r1 is correlated with r2 and r2 is correlated with r3. But the conflicting part between r2 and r3 is masked by r1 and will never be selected. So in reality, and if we consider only the applied parts in the firewall configuration, there is no conflict between r2 and r3. In this context, we define new conflicting rules-class anomalies:

3.2.1 Correlation

r_i and r_j are correlated if the intersection H of their domains is not empty and H is not shadowed by previous rules and the actions of both rules are different. Formally:
A rule $\{ri \Rightarrow A\}$ *is correlated with rule* $\{rj \Rightarrow \overline{A}\}$ $(j < i)$ *iff there exists at least a packet* $p \in \mathscr{P}$, *such that* $p \in dom(r_i)$ *and* $p \in dom(r_j)$ *and* $p \notin \cup_{k<j}\{dom(r_k)\}$.
For example, in Fig. 2, we can note that r_7 and r_8 are correlated. The conflicting flow between these two rules is the set A shown in Fig. 2.

3.2.2 Generalization

r_j generalizes r_i if the domain of r_i is included in the domain of r_j and the actions of both rules are different and r_i is not totally shadowed by its previous rules. Formally:
$\{rj \Rightarrow \overline{A}\}$ generalizes *A rule* $\{ri \Rightarrow A\}$ *iff for all packets* $p \in \mathscr{P}$, *if* $p \in dom(r_i)$ *then* $p \in dom(r_j)$ *with* $i < j$ *such that* $p \notin \cup_{k<i}\{dom(r_k)\}$.
For example, in Fig. 2, we can note that r_9 generalizes r_4. The conflicting flow between these two rules is the set B shown in Fig. 2.

4 Inference Systems for Discovering and Resolving Firewall Misconfigurations

In this section, we propose our approach as inference systems for discovering and resolving the two classes of firewall configuration anomalies presented in previous section.

4.1 Inference system for analyzing superfluous rules-class anomalies

In this section, we propose an approach to optimize a firewall configuration by detecting and removing all the superfluous rules defined in section 3. In fact, we propose our first inference system presented in Fig. 3. Its principle is as follows: The derived sets FC^{accept} and FC^{deny} are checked before and after removing each rule r_i in FR. If the sets remain unchanged, r_i is considered as superfluous and can be removed. FC represents the initial firewall configuration FR (before removing the superfluous rules).

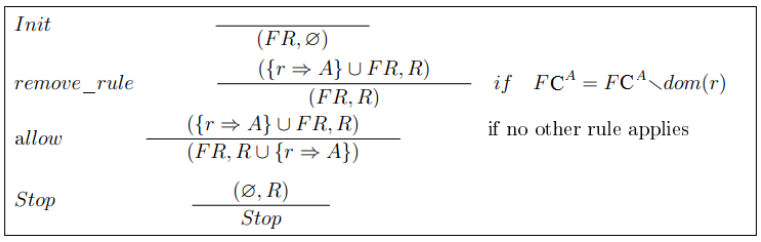


Figure 3: Inference system for the elimination of all superfluous rules

We write $C \vdash_{FR} C'$: C' is obtained from C by application of one of the inference rules of Fig. 3 (note that C' may be FR or updated version of FR which is R) and we denote by $\vdash^*_{FR}$ the reflexive and transitive closure of $\vdash_{FR}$.

Definition Let us consider FR_1 and FR_2 two distinct firewall configurations. We say that FR_1 and FR_2 are semantically equivalent if and only if $FR_1^A = FR_2^A$ for all $A \in \{accept, deny\}$.

Theorem If $(FR, \emptyset) \vdash^*_{FR} Stop$, then FR and R are semantically equivalent.

Proof If $(FR, \emptyset) \vdash^*_{FR} Stop$, then either all steps and not last one are follow. In such case, $R^A = FR^A$. Or either, at least the rule remove_rule applies on a rule $r \Rightarrow A$ where $FR^A = FR^A \setminus$ dom(ri). Let $D = \cup_i$ dom(ri). We can so show by induction on i that $R^A = FR^A \setminus dom(ri) = FR^A$. Therefore, FR and R are semantically equivalent.

4.2 Inference systems for analyzing conflicting rules-class anomalies

4.2.1 Discovering Conflicting rules-class anomalies

The rules of the system in Fig. 4 apply to quadruple $(FR, Anomalies, S, PC)$ whose first component FR is a sequence of filtering rules describing the firewall configuration and whose second component $Anomalies$ represents the list of anomalies detected and whose third component S is a subset of all packets filtered by anomalies-rules already detected. $Anomalies$ and S are respectively initialized to an empty set. The fourth component PC represents the preceding rules of a given rule $\{r_i \Rightarrow A_i\}$ in the firewall configuration. Detect_Ay is the main inference rule for the inference system shown in Fig. 4, it deals with the filtering rule $\{r_i \Rightarrow A_i\}$ with $A_i \in \{accept, deny\}$, of FR given in the quadruple. The condition for the application of Detect_Ay is that the intersection between the packets $dom(r_i)$ filtered by

$$Init \quad \frac{}{(FR, \varnothing, \varnothing, \varnothing)}$$

$$Define\_PC \quad \frac{(\{r_i \Rightarrow A_i\} \cup FR, Anomalies, S, \varnothing)}{(FR, Anomalies, S, PC)} \quad \text{where PC} = \{\{\mathrm{r}_k \Rightarrow \overline{A_k}\} \in \mathrm{R} \text{ with } \mathrm{k} < \mathrm{i}\ \}$$

$$\text{Detect\_Ay} \quad \frac{(FR, Anomalies, S, \{r_k \Rightarrow \overline{A_k}\} \cup PC)}{(FR, Anomalies \cup (\{r_i \Rightarrow A_i\}, \{r_k \Rightarrow \overline{A_k}\}, Rcorr), S \cup Rcorr, PC)} \; if(Dom(r_i) \cap Dom(r_k)) \setminus S \neq \varnothing)$$

$$Stop \quad \frac{(\varnothing, Anomalies, S, \varnothing)}{Stop}$$

Figure 4: Inference system for discovering conflicting rules-class anomalies

the rule r_i and packets $dom(r_k)$ filtered by a preceding rule r_k belonging to the previous configuration PC and not handled by S is not empty, i.e., we detect an anomaly (Correlation or Generalization) between the two rules r_i and r_k iff the intersection between these two rules is not totally masked by their preceding rules. So, S represents the set of conflicting packets of anomalies already detected. Applying this rule updates the list of anomalies by inserting a new anomaly represented by the triples $(r_i, r_k, Rcorr)$.
Theorem If $(FR, \emptyset, \emptyset, \emptyset) \vdash^*_{FR} Stop$, then all $Rcorr$ sets in $Anomalies$ tuples are disjoint.
Proof Suppose that $\exists\, Anomalies_i, Anomalies_j$ with $i < j$ such that $Rcorr_i \cap Rcorr_j \neq \emptyset$. In such case,$\exists$ $l < j$ such that $Rcorr_i \cap ((dom(r_j) \cap dom(r_l)) \setminus S_j) \neq \emptyset$. With $Sj = \cap_j Rcorr_j$. However, $R_i \subset S_j$. Thus, $Rcoor_i \cap ((dom(r_j) \cap dom(r_l)) \setminus S_j) = \emptyset$ Which is a contradiction. Therefore, we conclude that all Rcorr sets in $Anomalies$ tuples are distinct.

4.2.2 Resolving Conflicting rules-class anomalies

For each anomaly $(\{r_i \Rightarrow A_i\}, \{r_k \Rightarrow \overline{A_k}\}, Rcorr)$, we verify whether the conflicting part between r_i and r_k and not handled by all previous rules (i.e., $Rcorr$) have the same action in FR and in SP. If it is not the case, the conflict is considered to be misconfiguration and we should correct it. In other words, if we have a conflict between two rules and the conflicting part take the same action in SP and in FR, then, we should maintain our configuration file unchanged because this anomaly is intended. Thus, we define this notion which serves as foundation of our correction approach. The inference system in Fig. 5, contains three inference rules responsible for the whole process of the correctness of the firewall misconfigurations.
Correct is the main inference rule. It deals with the first anomaly $(\{r_i \Rightarrow A_i\}, \{r_k \Rightarrow \overline{A_k}\}, Rcorr)$ applying this rule updates the firewall configuration by inserting new rules Cr$(Dom(Rcorr) \cap (SP^{\{A_i\}}))$at the top of the FR while guaranteeing the same effect with respect to the conflict resolution, we can insert these rules at any position before r_k in the firewall configuration but we choose to insert them in the top of the sequence of rules to facilitate their identification by the administrator. The rules to insert are converted using the function Cr(p) with $p \in P$. This function converts each subset of packets into a finite sequence of rules that matches essentially these packets. Hence, successful repeated application of Correct ensures the correction of the firewall configuration with respect to the security policy. The stop rule in the inference system is applied when we parse all the anomalies of the list Anomalies.

The identification of misconfigurations is implicitly determined in our proposal. In fact, if $Rcorr \cap SP^{A_i}$ (with $\overline{A_k}$ is the action of r_k) is empty then we have not to insert new rules before r_k and we maintain our FR configuration unchangeable, so we implicitly mentioned here that the anomaly $(\{r_i \Rightarrow A_i\}, \{r_k \Rightarrow \overline{A_k}\}, Rcorr))$ is not a misconfiguration because the conflicting flow take the same action in SP and in FR.

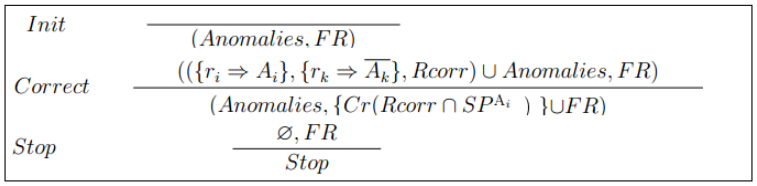


Figure 5: Inference system for resolving conflicting rules-class anomalies

Definition An anomaly is not considered to be a misconfiguration if and only if the action undertaken by the firewall FR for the conflicting part Rcorr is the same as defined by the security policy SP.
Theorem If $(Anomalies, FR) \vdash^{*}_{FR} Stop$ then all anomalies are not considered to be misconfigurations.
Proof If $Anomalies, FR \vdash^{*}_{FR} Stop$ then the inference rule correct is applied for each tuple anomaly Anomalies corresponding to a rule $r_i \Rightarrow A_i$. Let $C = Rcoor_i \cap SP^{A}$. When $C \neq \emptyset$, an anomaly Ay is considered as a misconfiguration since all packets belonging to C were in the process of undertaken a wrong action $\overline{A_i}$. As proved in theorem 2, the set $Rcorr_i$ is distinct from each conflicting domain $Rcorr_j$, with $j < i$. Applying the inference rule correct permits to add filtering rules matching the set C at the beginning of FR. By this way, the action A imposed by SP for packets in C is carried out. Therefore, successful repeated applications of the inference rule correct ensures that all anomalies are not considered to be misconfigurations.

5 Conclusion

In this paper we present our approach to classify, detect and correct firewall misconfigurations. The work presented provides essentially two mechanisms. First, we extract the misconfigurations in the firewall filtering rules, and second, we resolve these misconfigurations with respect to the requirement of the global security policy. The major contributions of the work can be stated as follows:

- The identification of all types of anomalies and the accurate designation of all rules involved in a given anomaly.
- The ability to make a distinction between intended anomalies and effective misconfigurations by considering the requirements of the security policy.
- The resolution approach is automatic since the security requirements are taken into account.

We believe that there is more to do in the firewall configuration management area. Our future research plan includes extending the proposed tool to detect and correct misconfigurations in a distributed environment. Our future work also includes optimizing proposed methods to correct misconfigurations.

References

[1] Alain Mayer, Avishai Wool, and Elisha Ziskind. fang: A firewall analysis engine. In Proceedings of 2000 IEEE Symposium on Security and Privacy, pages 177–187, 2000.

[2] Alex X. Liu. formal verification of firewall policies. In ICC, pages 1494–1498, 2008.

[3] athena firepac, 2012.

[4] P. Bera, S.K. Ghosh, and P. Dasgupta. policy based security analysis in enterprise networks: A formal approach. Network and Service Management, IEEE Transactions on, 7(4):231–243, 2010.

[5] Frédéric Cuppens, Nora Cuppens-Boulahia, and Joaquin Garcia Alfaro. detection and removal of firewall misconfiguration. In CNIS IASTED, Phoenix, AZ, USA novembre, 2005.

[6] Ehab S. Al-Shaer and Hazem H. Hamed. modeling and management of firewall policies. IEEE Transactions on Network and Service Management, 1(1):2–10, 2004.

[7] firewall builder, 2012.

[8] Hongxin Hu, Gail-Joon Ahn, and Ketan Kulkarni. detecting and resolving firewall policy anomalies. IEEE Transactions on Dependable and Secure Computing, 9(3):318–331, 2012.

[9] Hongxin Hu, Gail-Joon Ahn, and ketan Kulkarni. fame: a firewall anomaly management environment. In SafeConfig, pages 17–26. ACM, 2010.

[10] Bassam Khorchani, Sylvain Hall, and Roger Villemaire. firewall anomaly detection with a model checker for visibility logic. In NOMS, pages 466–469. IEEE, 2012.

[11] Muhammad Abedin, Syeda Nessa, Latifur Khan, and Bhavani M. Thuraisingham. detection and resolution of anomalies in firewall policy rules. In DBSec, pages 15–29, 2006.

[12] Naveen Mukkapati and Ch.V.Bhargavi. detecting policy anomalies in firewalls by relational algebra and raining 2d-box model. IJCSNS International Journal of Computer Science and Network Security, 13(5):94–99, 2013.

[13] Nihel Ben Youssef, Adel Bouhoula, and Florent Jacquemard. automatic verification of conformance of firewall configurations to security policies. In ISCC, pages 526–531, 2009.

[14] Padmalochan Bera, Santosh K. Ghosh, and Pallab Dasgupta. integrated security analysis framework for an enterprise network - a formal approach. IET Information Security, 4(4):283–300, 2010.

[15] Soutaro Matsumoto and Adel Bouhoula. automatic verification of firewall configuration with respect to security policy requirements. In CISIS, pages 123–130, 2008.

[16] Thawatchai Chomsiri and Chotipat Pornavalai. firewall rules analysis. In Security and Management, pages 213–219, 2006.

Author Index