List of Suggested Reviewers or Reviewers Not To Include (optional)

SUGGESTED REVIEWERS:
Not Listed

REVIEWERS NOT TO INCLUDE:
Not Listed

Collaborators and Other Affiliations List

Collaborators and Co-Editors:

Bjørner, Nikolaj (MSR Redmond); Gulwani, Sumit (MSR Redmond); Gvero, Tihomir (EPFL); Hillenbrand, Thomas (MPI-INF); Kloos, Johannes (MPI-SWS); Kuncak, Viktor (EPFL); Kuraj, Ivan (EPFL); Majumdar, Rupak (MPI-SWS); Mayer, Mikaël (EPFL); de Moura, Leonardo Mendonça (MSR Redmond); Niksic, Filip (MPI-SWS); Reinking, Alexander (UC Berkeley); Santolucito, Mark (Yale), Suter, Philippe (IBM Research); Waldmann, Uwe (MPI-INF); Weidenbach, Christoph (MPI-INF); Wies, Thomas (NYU); Yessenov, Kuat (MIT); Zhai, Ennan (Yale); Zufferey, Damien (MIT),.

Graduate and Postdoctoral Advisors

Kuncak, Viktor (EPFL)

Thesis Advisor and Postgraduate-Scholar Sponsor

Hallahan, Bill (Yale), ongoing.
Santolucito, Mark (Yale), ongoing.
Zhai, Ennan (Yale) ongoing.

Not for distribution

COVER SHEET FOR PROPOSAL TO THE NATIONAL SCIENCE FOUNDATION

PROGRAM ANNOUNCEMENT/SOLICITATION NO./DUE DATE NSF 16-578 11/16/16	☐ Special Exception to Deadline Date Policy	FOR NSF USE ONLY NSF PROPOSAL NUMBER
FOR CONSIDERATION BY NSF ORGANIZATION UNIT(S) (Indicate the most specific unit known, i.e. program, division, etc.) CCF - SOFTWARE & HARDWARE FOUNDATION		

DATE RECEIVED	NUMBER OF COPIES	DIVISION ASSIGNED	FUND CODE	DUNS# (Data Universal Numbering System)	FILE LOCATION
				043207562	

EMPLOYER IDENTIFICATION NUMBER (EIN) OR TAXPAYER IDENTIFICATION NUMBER (TIN) 060646973	SHOW PREVIOUS AWARD NO. IF THIS IS ☐ A RENEWAL ☐ AN ACCOMPLISHMENT-BASED RENEWAL	IS THIS PROPOSAL BEING SUBMITTED TO ANOTHER FEDERAL AGENCY? YES ☐ NO ☒ IF YES, LIST ACRONYM(S)

NAME OF ORGANIZATION TO WHICH AWARD SHOULD BE MADE Yale University	ADDRESS OF AWARDEE ORGANIZATION, INCLUDING 9 DIGIT ZIP CODE Yale University Office of Sponsored Projects New Haven, CT. 065208327
AWARDEE ORGANIZATION CODE (IF KNOWN) 0014266000	
NAME OF PRIMARY PLACE OF PERF Yale University	ADDRESS OF PRIMARY PLACE OF PERF, INCLUDING 9 DIGIT ZIP CODE Yale University AK Watson Lab New Haven ,CT ,065208285 ,US.

IS AWARDEE ORGANIZATION (Check All That Apply) (See GPG II.C For Definitions)	☐ SMALL BUSINESS ☐ FOR-PROFIT ORGANIZATION	☐ MINORITY BUSINESS ☐ WOMAN-OWNED BUSINESS	☐ IF THIS IS A PRELIMINARY PROPOSAL THEN CHECK HERE

TITLE OF PROPOSED PROJECT **SHF: Small: ConfigV: Automated Verification of Configuration Files**

REQUESTED AMOUNT	PROPOSED DURATION (1-60 MONTHS)	REQUESTED STARTING DATE	SHOW RELATED PRELIMINARY PROPOSAL NO. IF APPLICABLE
$ 499,295	36 months	06/01/17	

THIS PROPOSAL INCLUDES ANY OF THE ITEMS LISTED BELOW

- ☐ BEGINNING INVESTIGATOR (GPG I.G.2)
- ☐ DISCLOSURE OF LOBBYING ACTIVITIES (GPG II.C.1.e)
- ☐ PROPRIETARY & PRIVILEGED INFORMATION (GPG I.D, II.C.1.d)
- ☐ HISTORIC PLACES (GPG II.C.2.j)
- ☐ VERTEBRATE ANIMALS (GPG II.D.6) IACUC App. Date ________ PHS Animal Welfare Assurance Number ________
- ☒ FUNDING MECHANISM Research - other than RAPID or EAGER
- ☐ HUMAN SUBJECTS (GPG II.D.7) Human Subjects Assurance Number ________ Exemption Subsection ________ or IRB App. Date ________
- ☒ INTERNATIONAL ACTIVITIES: COUNTRY/COUNTRIES INVOLVED (GPG II.C.2.j) UK FR GM SP AS
- ☒ COLLABORATIVE STATUS Not a collaborative proposal

PI/PD DEPARTMENT Computer Science	PI/PD POSTAL ADDRESS AKWatson Hall 51 Prospect Street New Haven, CT 065208285 United States
PI/PD FAX NUMBER 203-432-0593	

NAMES (TYPED)	High Degree	Yr of Degree	Telephone Number	Email Address
PI/PD NAME Ruzica Piskac	PhD	2011	203-432-8001	ruzica.piskac@yale.edu
CO-PI/PD				
CO-PI/PD				
CO-PI/PD				
CO-PI/PD				

CERTIFICATION PAGE

Certification for Authorized Organizational Representative (or Equivalent) or Individual Applicant

By electronically signing and submitting this proposal, the Authorized Organizational Representative (AOR) or Individual Applicant is: (1) certifying that statements made herein are true and complete to the best of his/her knowledge; and (2) agreeing to accept the obligation to comply with NSF award terms and conditions if an award is made as a result of this application. Further, the applicant is hereby providing certifications regarding conflict of interest (when applicable), drug-free workplace, debarment and suspension, lobbying activities (see below), nondiscrimination, flood hazard insurance (when applicable), responsible conduct of research, organizational support, Federal tax obligations, unpaid Federal tax liability, and criminal convictions as set forth in the NSF Proposal & Award Policies & Procedures Guide,Part I: the Grant Proposal Guide (GPG). Willful provision of false information in this application and its supporting documents or in reports required under an ensuing award is a criminal offense (U.S. Code, Title 18, Section 1001).

Certification Regarding Conflict of Interest

The AOR is required to complete certifications stating that the organization has implemented and is enforcing a written policy on conflicts of interest (COI), consistent with the provisions of AAG Chapter IV.A.; that, to the best of his/her knowledge, all financial disclosures required by the conflict of interest policy were made; and that conflicts of interest, if any, were, or prior to the organization's expenditure of any funds under the award, will be, satisfactorily managed, reduced or eliminated in accordance with the organization's conflict of interest policy. Conflicts that cannot be satisfactorily managed, reduced or eliminated and research that proceeds without the imposition of conditions or restrictions when a conflict of interest exists, must be disclosed to NSF via use of the Notifications and Requests Module in FastLane.

Drug Free Work Place Certification

By electronically signing the Certification Pages, the Authorized Organizational Representative (or equivalent), is providing the Drug Free Work Place Certification contained in Exhibit II-3 of the Grant Proposal Guide.

Debarment and Suspension Certification

(If answer "yes", please provide explanation.)

Is the organization or its principals presently debarred, suspended, proposed for debarment, declared ineligible, or voluntarily excluded from covered transactions by any Federal department or agency? Yes ☐ [✓ No ☒]

By electronically signing the Certification Pages, the Authorized Organizational Representative (or equivalent) or Individual Applicant is providing the Debarment and Suspension Certification contained in Exhibit II-4 of the Grant Proposal Guide.

Certification Regarding Lobbying

This certification is required for an award of a Federal contract, grant, or cooperative agreement exceeding $100,000 and for an award of a Federal loan or a commitment providing for the United States to insure or guarantee a loan exceeding $150,000.

Certification for Contracts, Grants, Loans and Cooperative Agreements

The undersigned certifies, to the best of his or her knowledge and belief, that:
(1) No Federal appropriated funds have been paid or will be paid, by or on behalf of the undersigned, to any person for influencing or attempting to influence an officer or employee of any agency, a Member of Congress, an officer or employee of Congress, or an employee of a Member of Congress in connection with the awarding of any Federal contract, the making of any Federal grant, the making of any Federal loan, the entering into of any cooperative agreement, and the extension, continuation, renewal, amendment, or modification of any Federal contract, grant, loan, or cooperative agreement.
(2) If any funds other than Federal appropriated funds have been paid or will be paid to any person for influencing or attempting to influence an officer or employee of any agency, a Member of Congress, an officer or employee of Congress, or an employee of a Member of Congress in connection with this Federal contract, grant, loan, or cooperative agreement, the undersigned shall complete and submit Standard Form-LLL, ''Disclosure of Lobbying Activities,'' in accordance with its instructions.
(3) The undersigned shall require that the language of this certification be included in the award documents for all subawards at all tiers including subcontracts, subgrants, and contracts under grants, loans, and cooperative agreements and that all subrecipients shall certify and disclose accordingly.

This certification is a material representation of fact upon which reliance was placed when this transaction was made or entered into. Submission of this certification is a prerequisite for making or entering into this transaction imposed by section 1352, Title 31, U.S. Code. Any person who fails to file the required certification shall be subject to a civil penalty of not less than $10,000 and not more than $100,000 for each such failure.

Certification Regarding Nondiscrimination

By electronically signing the Certification Pages, the Authorized Organizational Representative (or equivalent) is providing the Certification Regarding Nondiscrimination contained in Exhibit II-6 of the Grant Proposal Guide.

Certification Regarding Flood Hazard Insurance

Two sections of the National Flood Insurance Act of 1968 (42 USC §4012a and §4106) bar Federal agencies from giving financial assistance for acquisition or construction purposes in any area identified by the Federal Emergency Management Agency (FEMA) as having special flood hazards unless the:
(1) community in which that area is located participates in the national flood insurance program; and
(2) building (and any related equipment) is covered by adequate flood insurance.

By electronically signing the Certification Pages, the Authorized Organizational Representative (or equivalent) or Individual Applicant located in FEMA-designated special flood hazard areas is certifying that adequate flood insurance has been or will be obtained in the following situations:
(1) for NSF grants for the construction of a building or facility, regardless of the dollar amount of the grant; and
(2) for other NSF grants when more than $25,000 has been budgeted in the proposal for repair, alteration or improvement (construction) of a building or facility.

Certification Regarding Responsible Conduct of Research (RCR)
(This certification is not applicable to proposals for conferences, symposia, and workshops.)

By electronically signing the Certification Pages, the Authorized Organizational Representative is certifying that, in accordance with the NSF Proposal & Award Policies & Procedures Guide, Part II, Award & Administration Guide (AAG) Chapter IV.B., the institution has a plan in place to provide appropriate training and oversight in the responsible and ethical conduct of research to undergraduates, graduate students and postdoctoral researchers who will be supported by NSF to conduct research.
The AOR shall require that the language of this certification be included in any award documents for all subawards at all tiers.

CERTIFICATION PAGE - CONTINUED

Certification Regarding Organizational Support

By electronically signing the Certification Pages, the Authorized Organizational Representative (or equivalent) is certifying that there is organizational support for the proposal as required by Section 526 of the America COMPETES Reauthorization Act of 2010. This support extends to the portion of the proposal developed to satisfy the Broader Impacts Review Criterion as well as the Intellectual Merit Review Criterion, and any additional review criteria specified in the solicitation. Organizational support will be made available, as described in the proposal, in order to address the broader impacts and intellectual merit activities to be undertaken.

Certification Regarding Federal Tax Obligations

When the proposal exceeds $5,000,000, the Authorized Organizational Representative (or equivalent) is required to complete the following certification regarding Federal tax obligations.
By electronically signing the Certification pages, the Authorized Organizational Representative is certifying that, to the best of their knowledge and belief, the proposing organization:
(1) has filed all Federal tax returns required during the three years preceding this certification;
(2) has not been convicted of a criminal offense under the Internal Revenue Code of 1986; and
(3) has not, more than 90 days prior to this certification, been notified of any unpaid Federal tax assessment for which the liability remains unsatisfied, unless the assessment is the subject of an installment agreement or offer in compromise that has been approved by the Internal Revenue Service and is not in default, or the assessment is the subject of a non-frivolous administrative or judicial proceeding.

Certification Regarding Unpaid Federal Tax Liability

When the proposing organization is a corporation, the Authorized Organizational Representative (or equivalent) is required to complete the following certification regarding Federal Tax Liability:

By electronically signing the Certification Pages, the Authorized Organizational Representative (or equivalent) is certifying that the corporation has no unpaid Federal tax liability that has been assessed, for which all judicial and administrative remedies have been exhausted or lapsed, and that is not being paid in a timely manner pursuant to an agreement with the authority responsible for collecting the tax liability.

Certification Regarding Criminal Convictions

When the proposing organization is a corporation, the Authorized Organizational Representative (or equivalent) is required to complete the following certification regarding Criminal Convictions:

By electronically signing the Certification Pages, the Authorized Organizational Representative (or equivalent) is certifying that the corporation has not been convicted of a felony criminal violation under any Federal law within the 24 months preceding the date on which the certification is signed.

Certification Dual Use Research of Concern

By electronically signing the certification pages, the Authorized Organizational Representative is certifying that the organization will be or is in compliance with all aspects of the United States Government Policy for Institutional Oversight of Life Sciences Dual Use Research of Concern.

AUTHORIZED ORGANIZATIONAL REPRESENTATIVE	SIGNATURE	DATE
NAME		

TELEPHONE NUMBER	EMAIL ADDRESS	FAX NUMBER

PROJECT SUMMARY

Overview:

Configuration errors (also known as misconfigurations) have become one of the major causes of system failures, resulting in security vulnerabilities, application outages, and incorrect program executions. Configuration errors were reported to be the largest fraction of failures in storage systems in a 2014 industry survey. Although many techniques have been proposed for a configuration error detection, these approaches mainly can be applied after an error has occurred.

We propose a framework for automated verification of configuration files. Verifying configuration files is different than verifying standard programs: configuration files are missing a program structure, but also a specification. They are written in a very low level programming paradigm: they mainly consist of sequences of assignments of some values to system variables. In such framework, writing a specification is complex task, since it is not clear how should the specification look like and which properties should it address. Our framework works as follows: in the pre-processing stage, we first automatically derive a specification. Once we have a specification, we check if a given configuration file adheres to that specification. Deriving a specification is based on analyzing a large number of configuration files from publicly available datasets. We recognize patterns and keyword correlations and from those findings we derive a corresponding specification. To do that we first develop a language model for configuration files. One question we need to address is: what if there is an error in the given training set? Our proposed framework is general enough that it can still derive a correct specification, under the assumption that the same error does not appear in a significant amount of files. We plan to evaluate our framework on real-world configuration files used in the TravisIC tool.

Intellectual Merit:

This work's key intellectual contributions are:

- A design of a new framework that can learn a language model from an only partially correct set of configuration files
- Investigating methods to handle the untyped and unstructured nature of configuration files
- Finding techniques to optimize learning parameters to handle the probabilistic nature of the training set
- Developing a modular tool for further work in configuration verification

Broader Impacts:

Our main goal is to help to reduce large scale system failures without adding any additional burden to users and system developers. Making progress towards verification of configuration files, will enable a static analysis of configuration files. This way potentially disastrous errors can be detected before they appear. We envision ConfigV to complement already existing post-failure error diagnosis efforts. In addition, this work will also benefit other verification domains, such as network traffic verification.

TABLE OF CONTENTS

For font size and page formatting specifications, see GPG section II.B.2.

*Proposers may select any numbering mechanism for the proposal. The entire proposal however, must be paginated. Complete both columns only if the proposal is numbered consecutively.

1 Motivation

Configuration errors are one of the most important root causes of today's software system failures [30, 31]. Recently, there was a problem in accessing Facebook and Instagram [1], and a Facebook spokeswoman reported that this was caused by a change to the site's configuration systems. In a recent software system failures study [31], researchers revealed that about 31% of system failures were caused by configuration errors, which is higher than the percentage of failures resulting from program bugs (20%). Configuration errors are commonly referred to in the literature as misconfigurations.

Misconfigurations, in practice, may result in various system-wide problems, such as security vulnerabilities, application crashes, severe disruptions in software functionality, and incorrect program executions [33, 32, 29, 27].

The systems research community has recognized this as an important problem. In fact, at this year's OSDI conference (Operating Systems Design and Implementation, a top-tier system conference) a paper on detecting configuration errors by emulating the late execution [28] received the best paper award. While many efforts have been proposed to check, troubleshoot, diagnose, and repair configuration errors [7, 24, 26], those tools mainly try to understand *what* caused the error – they are still not on a level of automatic verification tools used for regular program verification [17, 20, 10] that can detect errors without executing the code. Our proposal offers automated verification of configuration files: we are analyzing files and proactively reporting potential errors, without waiting for them to happen.

1.1 Examples

We start by presenting two non-trivial configuration errors extracted from real-world examples. Although the errors are relatively simple, we call them non-trivial, because the majority of existing misconfiguration checking tools [33, 25] cannot detect these configuration errors. Most of the presented examples were found on StackOverflow [4], a popular Q&A website for programmers.

Example 1: Ordering errors. When a user configures PHP to run with the Apache HTTP Server, most likely the user will take some already existing configuration files and adapt them to suit her needs. The configuration file might contain, among others, the following lines:

```
extension = mysql.so
...
extension = recode.so
```

In this case, the configuration file will cause the Apache server to fail to start due to a segmentation fault error. This is because, when using PHP in Apache, the extension `mysql.so` depends on `recode.so`, and their relative ordering is crucial. This is an example of an *ordering error*. Yin *et al.* report that ordering errors widely exist in many system configurations, *e.g.*, PHP and MySQL, and typically lead to multiple system crash events. However, no existing tool in the systems research area can effectively solve or detect this problem [33, 30, 29].

Example 2: Fine-grained value correlation error. Our next example also comes from a discussion on StackOverflow [2]. The user has configured her MySQL as follows:

```
key_buffer_size = 384M
max_heap_table_size = 128M
max_connections = 64
thread_cache_size = 8
...
sort_buffer_size = 32M
join_buffer_size = 32M
read_buffer_size = 32M
read_rnd_buffer_size = 8M
...
```

The user complained that her MySQL load was very high, causing the website's response speed to be very slow. In this case, `key_buffer_size` is used by all the threads cooperatively, while `join_buffer` and `sort_buffer` are created by each thread for private use; thus, the maximum amount of used key buffer, *i.e.*, `key_buffer_size`, should be larger than `join|sort_buffer_size` * `max_connections`. Clearly, in the above example, this does not hold, so this misconfiguration causes MySQL to load very slowly. This type of error is more sophisticated than the simple value correlation that some tools can detect [31, 33].

1.2 Challenges

We believe there are two main obstacles to simply applying the existing automatic tools and techniques to verification of configuration files:

- the lack of a specification describing properties of configuration files
- the structure of configuration files – they are mainly a sequence of entries assigning some value to system variables (called *keywords*).

The language in which configuration files are written does not adhere to a specific grammar or syntax. In particular, the entries in configuration files are untyped. Moreover, there are surprisingly few rules specifying constraints on entries, and there is no explicit structure policy for the entries. We believe these to be the reason for the lack of automated verification of configuration files,despite the systems community desires for such verification [25, 33, 30].

1.3 Automated Verification of Configuration Files

The goal of this proposal is to develop a fully **automated verification framework for general software configurations**. We plan to overcome the above obstacles by first automatically inferring a specification for configuration files. It is unrealistic to expect the users to write an entire specifications for configuration files on their own. This process can easily lead to incomplete or even contradictory specifications. Instead, we will learn specification from a large sample of configuration files,which will be taken by the learning process as input. The process will be language-agnostic and should work for any kind of configuration files, but all of the files in the sample need to be of the same kind (such as MySQL or HTTPD configuration files). From that sample, we will learn an abundant set of rules specifying various properties that hold on the given sample. The rules, in general, specify which properties keywords in configuration files need to satisfy. One can see this learning process as a way of deriving a specification for configuration files. It is hard to talk here about a complete specification, but still having some specification is better than none, and we plan to increase the preciseness of a specification as we gain a better understanding of the formalism needed for its description. With a specification, we can efficiently check the correctness of the configuration files of interest and detect potential errors. Errors are reported if the configuration file does not adhere to the derived specification.

For practical purposes we plan to use a real-world dataset [3]. The files in the sample might contain errors, but they are typically different errors and only appear in a small percentage of files. We will, therefore, use probabilistic learning to derive a set of accurate rules.

Building such an automatic verification framework for configuration files requires addressing several challenges. First, in the process of inferring a specification, we have to analyze mainly with an untyped, unstructured sequence of assignments. Thus, we need to develop a suitable language model. We do that by "guessing" a type of keywords and then deriving formulas that describe relationships between these keywords. However, the type of a keyword cannot always be fully determined from a single value. An entry `general_log = 1` in a MySQL configuration file assigns an integer to the keyword `general_log`. However, 1 here is a Boolean variable, which denotes that there will be a general log file. Another example, which often appears in practice, shows how keyword names and their "types" can easily lead to an error. Consider the entry `general_log = /var/log/mysql/mysql.log`. Although this statement alone might look correct, this is actually a typing error which will prevent the MySQL log from being correctly written.

Second, we will learn patterns describing keywords and their relations. We will associate with each type a set of very general templates. The user does not need to provide any templates – they are internally associated to the types. In addition to those general templates, we still need specific algorithms to learn rules that cannot be easily templated (such as ordering rules).

We will implement this verification framework and call it ConfigV (for continuity reasons – our preliminary tool was called ConfigC, cf. next section).

2 Our Previous Work: ConfigC

Based on our own experience with misconfigurations, we have proposed and developed a simple tool for verification of configuration files, named ConfigC [23]. Motivated by promising preliminary results, which we obtained with ConfigC, we believe that automated verification of configuration files is possible and that it should be further studied.

From ConfigC we drew the main inspiration for this proposal. We now briefly outline how ConfigC works. It first automatically analyzes a dataset of correct

```
File_1:
max_connections=300
general_log=1
...
mysql.max_persistent=200
```

```
File_2:
max_connections=400
general_log=0
...
mysql.max_persistent=200
```

Figure 1: Fragments of two configuration files for configuring PHP on MySQL

configuration files to derive rules for describing a specification. Before that process even starts, we identified several basic types, such Boolean, Integer, File. With each of these types we associated a list of templates. We then analyze the given training data set of correct configuration files and derive rules describing variable relations. We illustrate how is this done on two fragments of configuration files given in Fig. 1. For simplicity, let us assume that all visible keywords are integers. With the integer type we only associated comparison templates ($X < Y, X = Y, \ldots$). After analyzing File_1, the set of learned rules will contain the following rules:

```
max_connections=300
mysql.max_persistent=200
max_connections > mysql.max_persistent
```

The set of learned rules changes with every analyzed file: contradicting rules are removed and new rules are added. After analyzing File_2 the rule `max_connections = 300` is removed. In practice, this process is done in parallel.

Additionally, we also developed algorithms for misconfigurations that cannot easily be templated. All together, ConfigC can detect the following errors: missing entries, ordering errors, type errors and simple value correlations. Once this preprocessing of training data is done, the resulting language model, together with accompanying rules, is used to detect errors in new configuration files given by the user.

Running ConfigC on Example 1 from Sec. 1.1 returns the following output:

```
ORDERING ERROR: Expected extension "recode.so" BEFORE
extension "mysql.so"
```

ConfigC could not detect the error present in Example 2 since it can detect only simple comparison-based relations.

We implemented a prototype and evaluated it on real-world MySQL configuration files [3]. We manually extracted 20 files that contained errors which our tool can detect and grouped them into four categories according to their error types: missing entry, type error, ordering error and value correlation. The evaluation results are shown in Table 1. We can see that the our tool can detect most of the errors, with a low number of false positives. We believe that further studies in this direction will bring us better results

Table 1: Evaluation results on misconfiguration detection of ConfigC

Error Type	Passing Tests	False Positives
Missing Entry	5/5	1, 0, 0, 0, 4
Type Error	5/5	0, 0, 0, 0, 0
Ordering Error	5/5	0, 2, 1, 0, 6
Value Correlation	4/5	0, 0, 0, 1, 0

2.1 Limitations of ConfigC

Despite promising early results, ConfigC was a first experimental attempt into formal verification of configuration files, and as such it still has several limitations.

1. ConfigC works on the premise that the training dataset consists only of correct files, which is hard to achieve in practice. In the prototype we manually checked every file in the prototype. Clearly, this approach will not scale. There exist a large set of real-world misconfigurations [3]. Nevertheless, the errors are distributed through the files and do not all appear at the same spot. We believe that it should be possible to construct a correct specification from the rules that hold in a significant majority of the files.
2. The language model used in ConfigC cannot handle a simple version of the `if-then` command that appears in some configuration files (such as Apache HTTPD). We already pointed out some issues with parsing and possible incorrect typing. We need to develop a more accurate model which will be a uniform representation for diverse configuration files.
3. ConfigC offers only a limited number of templates to to detect correlation errors on keywords. It can detect, for example, keyword comparisons, but in practice many software outages are caused by more complex correlations (also called fine-grained correlations [2]). As an illustration, ConfigC cannot find an error in Example 2 in Sec. 1.1. However, to detect more complex rules, we will need to use tools like SMT solvers [12, 8].
4. ConfigC cannot check for anomalies in a value set for one particular keyword. For example, setting a parameter, which relates memory in MySQL configuration files, too large can exhaust the RAM, leading to extreme slowness or even a crash. Such an anomalous value does not violate any previously mentioned constraints (such as type errors or ordering errors), but it makes system behave incorrectly.

This proposal aims to address all the above limitations.

3 The ConfigV Framework Overview

We propose to build ConfigV, an automatic verification framework for software configuration files, by exploring further the insights we obtained when building ConfigC. Figure 2 presents a typical ConfigV verification workflow which contains three steps: translation into an intermediary representation, learning of a specification, and checking the correctness of a given configuration file.

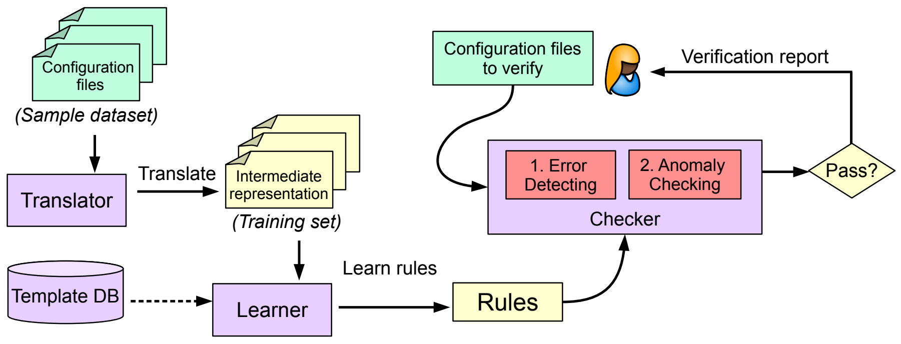

Figure 2: ConfigV's workflow. The green components represent configuration files, including both sample configuration datasets and users' input configuration files to verify. The purple components are the modules of ConfigV. Because template DB is not necessarily used, we use dashed arrow between it and the learner. Red boxes are sub-modules within the checker. The yellow components are results generated by ConfigV's modules.

Learning Specifications. As with ConfigC, we start with an assumption that we are given a number of configuration files belonging to the same system (*e.g.*, MySQL or Apache). Additionally, we assume that files in this dataset are not necessarily correct, but if there are errors they appear only in some files. We then translate the dataset into a more structured and typed intermediate representation, which follows our defined language model (cf. Sec. 3.1). Since the type of an entry cannot be fully determined from the entry value, we introduce *probabilistic types*, which are a more flexible type notion: to an indefinite keyword it assigns a list of possible types, along with their probability distribution. The learner module employs a collection of learning algorithms to generate various rules and constraints. It uses type-annotated templates to infer the rules. In addition, there are algorithms for detecting ordering errors and missing entry errors, which do not depend on templates. However, this time, since the input set is not necessarily correct, the learned rules are also annotated with the probability indicating the likelihood of being correct.

Correctness Checking. The checker takes as input a configuration file and reports if the file adheres to the learned specification. In addition, it will also check that the keyword values fall into a certain range and they are not significantly different than the value usually used. The checker generates a report (as shown in Sec. 2) about found errors.

3.1 A Language Model for Configuration Files

There are several types of configuration files, and each has a different representation. Our goal is to find a good intermediary representation that will be language-agnostic. In general, one could argue that the design of configuration files is pretty low level, and that we need a new programming language for them. We agree with that assertion, and actually our intermediary representation can be seen as a new programming language for writing configuration files, but this is not the main focus of this proposal. Previously, Huang *et al.* [15] proposed a language, ConfValley, to validate whether given configuration files meet administrators' specifications. However, ConfValley cannot check for misconfiguration checking capability, since it is only a library. In ConfValley the administrator still needs to write specifications manually, which is an error-prone process. On the contrary, ConfigV does not require users to manually write anything.

The translator is a part of ConfigV. It takes as input a sample dataset of configuration files and transforms it into another set of files written in a typed and well-structured form. The translator can be seen a parser, and it is used to translate files into an intermediate representation.

Translating or parsing is system dependent. In other words, for MySQL and HDFS, we need to develop different parsers. Configuration files mainly consists of assignments of the form `keyword = value`. Our first attempt was to simply translate every key-value entry `k = v` into a triple (k, v, τ), where τ is a type of v. However, sometimes we could not fully determine the type of key based on a single example value. For this reason, we introduced *probabilistic types*.

Consider the following example.

```
foo = 300
bar = 300.txt
```

Most likely `foo` should be an integer, but it could also be a string. In the second case, we can learn the rule stating `foo` $\in$ $\mathsf{substrings}$(`bar`). Instead of assigning one type to a value, the translator assigns a distribution of types to a value, an idea closely related to existentially quantified types [16].

Formally, we define probabilistic types as follows: let $\mathcal{T}$ be a set of basic types (cf. Table 2). A probabilistic type built from $\mathcal{T}$ is a list of pairs $[(\tau_1, p_1), \ldots, (\tau_n, p_n)]$, such that $\tau_i \in \mathcal{T}$, $0 \leq p_i \leq 1$ and $\Sigma p_i = 1$. These probabilities are updated each time a new example value for a key is encountered.

When a value has a probabilistic type, we generate rules for all its types. This means that by assigning `foo` a probabilistic type (*e.g.*, `(foo, 300, [(`*`Int`*`,90%), (`*`String`*`,10%)])`, we would generate rules for both strings and integers. Once the type inference can uniquely determine the type, the probability of all other types is set to zero, and the associated rules are withdrawn. With the help of probabilistic types, we addressed ambiguities during the parsing phase.

In ConfigV, the set $\mathcal{T}$ should contain strings, integers, file paths, sizes, and IP addresses. With every type, we associate a list of templates that are used to learn the rules about those files. Table 2 contains the most important types and templates.

Table 2: Table of types, along with associated templates, used in ConfigV

Type	Template
Integer	$X = Y, X \neq Y, X \leq Y, X < Y, X * Y < Z, \ldots$
String	$\mathsf{substr}(U, V), \mathsf{prefix}(U, V), \mathsf{suffix}(U, V), \ldots$
File Path	$\mathsf{isFile}(F), \mathsf{isDir}(D), \ldots$
Size	Similar to integers
Port	$P_1 > P_2$
IP Addr.	$\mathsf{sameSubnet}(addr1, addr2)$

In general, the templates works as follows: if there are two entries `k1` = `v1` and `k2` = `v2`, such that $t(X, Y)$ is a template that can be applied to those values (w.r.t. their types). If $t(v1, v2)$ holds, *i.e.*, the template condition is valid for those concrete values, then we will add a rule $t(k1, k2)$ to a list of potentially correct specifications. Note that the rule expresses a relation between keywords, and not values.

Another problem that we need to address is that configuration files sometimes contain a simple version of the `if-then`, such as in Apache HTTPD, that sets conditions for which an action should be applied. Therefore, we also add this guard g to our entry during the parsing phase.

With guards in the picture, when instantiating the rules, we first check if the guard is true. If it is not, we simply skip that entry.

To summarize, the translator translates every entry `k` = `v` in a configuration file into a quadruple entry

$$(g, k, v, \mathsf{List}[p_1 : \tau_1, \ldots, p_n : \tau_n])$$

4 The Learner Module

The goal of the learner module is to derive rules and constraints from the intermediate representation generated by the translator. In general, the learner module has two components. The first component (Sec. 4.1) learns rules for checking configuration errors like missing entry errors, ordering errors, and fine-grained value correlation errors. These errors tend to cause total system failures. The second component (Sec. 4.2) aims to derive constraints on entries to check for suspicious (or anomalous) values that may violate standard practice. These anomalies can cause partial degradation of the system, such as significant reduction in performance, or even total failure [33].

The approach in our earlier work [23] is guaranteed to produced only correct rules. As opposed to a traditional machine learning approach, we have here a formal guarantee of rule coverage. Furthermore, by design this approach gives a justification (error location) for the classification of a user file as correct or misconfiguration. This approach is most closely related to version space learning [19], which we discuss further in Sec. 5.

4.1 Derivation of ConfigV Rules

In ConfigV's learner module we need to develop a learning mechanism that is tolerant enough to accept a dataset of possibly incorrect configuration files. Rather than manually correcting each file, we extend the ConfigC formalism to run probabilistic learning on the intermediate representations. Our probabilistic approaches for learning missing entry, ordering, and fine-grained value correlation rules stem from existing work with building non-probabilistic versions of these rule-learning algorithms. For each of these rules, we consider all possible permutations of keys that appear in every file which are appropriately typed, and for our learning process, calculate the likelihood that each of these permutations constitute a rule. Broadly speaking, we accept rule patterns that appear frequently enough.

We now describe a formalism how to compute the set of learned probabilistic rules, denoted by $\mathcal{PR}$. With $\mathcal{U}$ we denote a set of configuration files in the intermediate representation. If a template can be instantiated with keywords from file f and the obtained rule r holds, we state that with the predicate $\mathit{appears}(r, f)$. If a template is instantiated and we obtain rule r, with the predicate $\mathit{holds}(r, f)$ we state that either predicate $\mathit{appears}(r, f)$ holds, or r could not be instantiated with keywords from f (namely, rule r is irrelevant for f and then trivially holds). With every rule r we define two sets:

$$S_A(r) = \{\ f \in \mathcal{U} \mid \mathit{appears}(r, f)\ \}$$
$$S_H(r) = \{\ f \in \mathcal{U} \mid \mathit{holds}(r, f)\ \}$$

We define then set $\mathcal{PR}$ as

$$\mathcal{PR} = \{\ r \ \mid\ |S_A(r)| > T_1 \wedge |S_H(r)|/|\mathcal{U}| > T_2\ \}$$

We use T_1 to act as an integer threshold value so any accepted rule must appear in sufficiently enough files where the rule appears. We use T_2 to act as a percentage threshold value so that any accepted rule must be true in a majority of cases. Finding the exact values of T_1 and T_2 and their connection to $|\mathcal{U}|$ is a question still to be solved. Even taking a simple majority voting approach, we must find the cutoff value that will yield the best results in learning. We plan to try multiple strategies to find these values. One approach may be to use machine learning for program autotuning, with a tool like OpenTuner [5]. This requires creating some cost function for which the autotuning can optimize. Additionally, it is possible that T_1 and T_2 differ depending on the quality of the learning set and/or the particular type of rule being learned. Building a clear formalization and a modular tool to explore these questions is major task.

Note that this approach differs from a unsupervised learning setting [14] where the training set is completely unlabeled and seeks to discover some hidden underlying structure. Although our set is also unlabeled, we make the assumption that some majority of the training set must be correct.We are again learning a model which will provide clear justification (error locations) for the classification. However, with this approach we will not have the formal guarantees of the ConfigC approach. From previous work, we believe this is a worthwhile trade-off in order to decrease the false positive rate.

4.2 Learning Deviated Constraints

In addition to checking for correlation, type, or ordering errors, the user may also want to examine if there are anomalous values that a keyword is assigned to. Such errors may also cause tricky and critical performance issues that are hard to debug. Consequently, anomalous values should be flagged and a warning returned to the user indicating the violation.

We now describe a technique that we plan to explore to detect anomalous values for numerical attributes. Let A be the set of attributes contained in the configuration files in the sample dataset. Let A_n be the subset of attributes of A which are numerically typed. Then, for each attribute $a \in A_n$, we construct a vector v_a of the values corresponding to attribute a, seen over the entire sample dataset. For each v_a, we compute an interval

$$[\hat{v_a} - 50 * MAD(v_a), \hat{v_a} + 50 * MAD(v_a)],$$

where $\hat{v_a}$ represents the median over the values in v_a and $MAD(v_a)$ refers to the median absolute deviation. This is a variant of a standard outlier detection test, namely the Hampel identifier. (Mathematically, $MAD(v_a) = 1.4826 * median(|v_a - \hat{v_a}|)$, estimating standard deviation for a normal distribution.) In the checking phase, if the checker finds a value for a numerical attribute in the checked file outside of this interval, a warning would be printed to the user indicating the violating value, the attribute, and the upper or lower Hampel threshold.

The intuition behind this is that if the user has input a value that falls outside of an interval containing values that are considered “normal” over the entire sample dataset, that value will probably cause an error, in particular for performance. We cannot know for sure if this value will cause an issue. For instance, a user might have a machine with particularly high-end hardware, in which case a value beyond the upper Hampel threshold may be appropriate.

5 Extending ConfigV and Applying it to Real-World Configurations

TravisCI [9] is a popular integration and build testing service connected to Github. It has been used on more than 17 million projects. TravisCI allows programmers to automatically run their test suite on a fresh virtual machine every time they commit their code. A TravisCI user must add a configuration file to the repository that specifies build conditions, such as which dependencies are required, and a set of benchmarks to test. The results of these tests (indicating either passing, failing, or misconfiguration) are saved by TravisCI on a publicly available database [9].

TravisCI misconfigurations are important to detect and correct before execution, because any failed compilation can costs developer significant amount of time. A misconfiguration can cause a test suite to run up to 20 minutes, only to report that the whole computation failed [9]. If a developer had this information prior to the compilation attempt, they could identifying and correct potential misconfigurations.

A new challenge here is that we need to reason about a temporal component too. Having an entire TravisCI log for a given project we can examine the sequence of changes of a configuration file and detect what has changed, so that we switched from a failing compilation to successful compilation.

5.1 Version Space Learning with Temporal Properties

We plan to use version space learning and SMT solvers to detect errors in TravisCI configurations files. Version space learning builds a logical constraint model for binary classification [19], which we use to test a configuration file for membership in the set of all correct files.

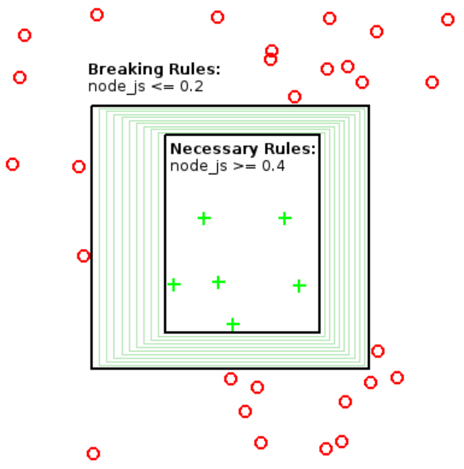

Figure 3: Red circles are failing files, and green pluses are passing files.

Figure 3 outlines our proposed approach. We first apply the standard ConfigC approach to learn the rules that have to hold for correct TravisCI configuration files. Traditionally, version space learning builds a model that defines membership using a series of disjunctions from a set of predefined hypotheses. However, we need to use conjunctions to convey that a configuration file is only correct if it satisfies all the learned relations. Those rules can be seen as positive examples in classifying correct files. In addition, we will have so-called *breaking rules* which are derived from the files that do not compile, *i.e.* negative examples. Breaking rules are specified using disjunctions.

Figure 3 shows an example set of necessary and breaking rules that would be learned if the training set contains a failing file indicating that node_js version is 0.2, and a passing file has its version set to 0.4. With red dots we depict incorrect files that do not compile, and the green crosses show correct files. From both types of files, we extract breaking and necessary rules, which define borders between those two sets of files. This results in a space (green stripes) where a file may contain relations that cannot be classified as either breaking or necessary. For example, if a user provides a file which specifies version 0.3 of node_js, this file will fall in the green striped space. Part of the research task will be to find an optimal classification for such cases.

The success or failure of a TravisCI configuration file is dependent not only on the configuration file itself (namely, a .travis.yml file) but it also depends on system information such as programming languages and a list of imported libraries. For this reason, we call this extended configuration files a program summary, denoted with P_t. This is a representation of the repository which contains any information relevant to the learning process. The subscript on P_t is a timestamp tag based on the ordered git commit history.

From these summaries we will build a model $M(P_t)$, using the same techniques as in ConfigV. $M(P_t)$ actually is a set of all possible relations derivable from the program summary. As previously, we need to construct templates for the learning process, since the level of specification details depends on them.

We now outline how we plan to use SMT solvers to find the sets of necessary rules (Nec), and the set of breaking rules (Br).

Let $S(P_t)$ be the build status returned from TravisCI when run on P_t. It can be either $Pass$ or Err. We define shorthand $S(P_{t,t+1}) = PE$ (also called a break state), to express that the program was compiling before and now it does not:

$$S(P_{t,t+1}) = PE :\Leftrightarrow S(P_t) = Pass \wedge S(P_{t+1}) = Err$$

If executing a TravisCI file fails, then there must exist at least one rule that is causing that error. Similarly, if a build is passing, then there must not exist any error. That is, the model of a passing commit must not contain any breaking rules. Note we are not, however, guaranteed that any rules from a passing commit are necessary.

$$S(P_t) = Err \Rightarrow \exists r \in M(P_t), r \in Br \tag{1}$$

$$S(P_t) = Pass \Rightarrow \forall r \in M(P_t), r \notin Br \tag{2}$$

Additionally, when we commit a break (PE), we can localize the error to one of the relations that changed. Either we removed something that was necessary, or added something that was breaking. Note that this is an inclusive disjunction, since a erroring commit can break multiple things at once. Expressed formally, where \ is the set difference, that is:

$$\begin{aligned} S(P_{t,t+1}) = PE \Rightarrow & \\ & \exists r \in (M(P_t) \ \setminus M(P_{t+1})), r \in Nec \\ & \vee \exists r \in (M(P_{t+1}) \setminus M(P_t)), r \in Br \end{aligned} \tag{3}$$

We then create a conjunction of (1), (2), and (3) and invoke an SMT solver to find a model for that formula. The resulting model will define sets of necessary rules (Nec), and the set of breaking rules (Br), which can be used to check new configuration files. Since we used a similar model to ConfigV, we will still be able to provide justifications for the classification results.

We also believe that this approach will work in practice since travis.yml files are relatively small and changes in the files are often at most 1-2 lines.

6 Related Work

We will now compare our work to existing work. Most of this work falls into what is traditionally considered system's research.

Configuration languages. There have been several language support efforts proposed for preventing configuration errors introduced by fundamental deficiencies in either untyped or low-level languages. For example, in the network configuration management area, it is easy for administrators to produce configuration errors in their routing configuration files. PRESTO [13] automates the generation of device-native configurations with configlets in a template language. Loo *et al.* [18] adopt Datalog to reason about routing protocols in a declarative fashion. COOLAID [11] constructs a language to describe domain knowledge about devices and services for convenient network reasoning and management.

Compared with these existing efforts, our work is mainly focused on software systems, *e.g.*, MySQL and Apache, rather than network configurations. In addition, we do not need the user of ConfigV to manually write a configuration file with the proposed language, since ConfigV can automatically parse a target configuration file into our proposed representation.

We previously discussed ConfValley by Huang *et al.*, which proposed a specification for configuration validation [15].

Misconfiguration detection. Misconfiguration detection techniques aim at checking configuration efforts before system outages occur. Most existing detection approaches check the configuration files against a set of predefined correctness rules, named constraints, and then report errors if the checked configuration files do not satisfy these rules.

Several machine learning-based misconfiguration detection efforts have been proposed [32, 33]. EnCore [33] is the work closest to ConfigV. It introduces a template-based learning approach to improve the accuracy of their learning results. The learning process is guided by a set of predefined rule templates that force learning to focus on patterns of interest. In this way, EnCore filters out irrelevant information and reduces false positives; moreover, the templates are able to express system environment information that other machine learning techniques cannot handle. Compared to EnCore, ConfigV has the following advantages. Firstly, ConfigV does not rely on 100% correctness in the files of the given configuration set. Secondly, ConfigV not only covers many more types of misconfigurations, but also introduces probabilistic types. Finally, ConfigV is a language framework, which can even be used to write configuration files, whereas EnCore is only a misconfiguration detection tool.

Misconfiguration diagnosis. Many misconfiguration diagnosis approaches have been proposed [7, 6]. For example, ConfAid [7] and X-ray [6] use dynamic information flow tracking to find possible configuration errors that may result in failures or performance problems. AutoBash [24] tracks causality and automatically fixes misconfigurations. Unlike ConfigV, most misconfiguration diagnosis efforts aim at finding errors after system failures occur, which leads to prolonged recovery time.

Misconfiguration tolerance. There have been several efforts proposed to test whether systems are tolerant to misconfigurations [29]. SPEX [29] takes a white-box testing approach to automatically extract configuration parameter constraints from source code, and generates misconfigurations to test whether systems can tolerate potential configuration errors.

Making systems gracefully handle misconfigurations and eliminating configuration errors are two orthogonal directions. The former helps improve the robustness of systems and make diagnosis easier. This is especially important for software that will be widely distributed to end users. Our work belongs to the latter case, which is used to prevent configuration errors before system outages.

7 Project Timeline

The proposed project is supposed to last three years. We have identified three main goals of the project, each of them taking approximately a year:

Language Model Our language model has to be expressive enough to be able to express and detect many non-trivial misconfiguration cases. We will, therefore, need to define a more comprehensive language model (than the one presented). The configuration files of some of software systems, *e.g.*, squid, are not limited to the keyword-value parameters; thus, we need to extend the current language model.

Learning Algorithms We outlined basic learning algorithms. However, many details are left unspecified and finding the right thresholds will be one of the main tasks. We plan to investigate some existing learning algorithms for programs, *e.g.*, Raychev *et al.* [22, 21]. The configuration files have a specific structure and directly applying existing rules will not be possible. Still we will try to adapt some of these algorithms and ideas to our problem.

Real-Wolrd Evaluation Finally, we will implement and evaluate ConfigV on real world misconfigurations. In addition to evaluating ConfigV on TravisCI files, during the development process we will empirically evaluate efficiency, usability, and accuracy of ConfigV. We plan to use the existing publicly available dataset of misconfigurations [3]. The set is large enough to use some parts as a training data and some parts for evaluation.

8 Broader Impact

Since her beginning of teaching at Yale, the PI is actively including topics from her research into her courses (such as “Software Analysis and Verification”). This way she motivates undergraduate students to get involved in the research. Until now she supervised more than 20 undergraduate student projects. Two students (Reinking, Cai) who did an undergraduate research projects with her, are now PhD students at UC Berkeley, and two more students are applying (Anklessaria, La) to graduate schools this year. Additionally, there are two student startups, which are now supported by YCombinator, that started under her supervision. Those startup are PatientBank and Py, an app for learning Python.

The PI will continue to involve the undergraduate student into this project as well, spreading this way the project ideas to a broader community. In addition, we plan to disseminate the results via publications and giving talks to the communities of PL/Verification researchers and System researchers.

We believe that ConfigV could have the following broader impacts:

- **Enabling language-based configuration verification.** Today's misconfiguration checking approaches are still using *ad hoc* algorithms to detect potential misconfiguration root causes. ConfigV should enable more rigorous verification of configuration files. By verification we mean checking that the given configuration file adheres to a specification.
- **Complementing post-failure error diagnosis efforts.** There is a number of diagnosis and troubleshooting efforts to localize the root causes of configuration errors after system failures occur. ConfigV's approach of pre-failure error checking can be seen as a preliminary checking before installing a system. Because ConfigV can detect many non-trivial configuration errors, it simplifies work for post-failure error diagnosis tools. Additionally, the output results of ConfigV can be used as a guidance for post-failure error diagnosis approaches, making them more efficient.
- **Benefiting other communities, *e.g.*, network misconfiguration.** Besides software misconfiguration, network community has the same misconfiguration problems. The ideas presented here could be extended to reason about network-level configuration errors.

9 Prior Results from NSF Support

The PI received NSF CAREER Award for synthesis in live programming environment (CCF-1553168). In addition, the PI received some student travel grants to support students' visit to conferences or summer schools. The most relevant NSF support for this proposal is the NSF award "SHF: Medium: Collaborative Research: FRP for Real", (CCF-1302327); $866,000 (10/01/13-09/30/17), which the PI overtook from Paul Hudak, due to his untimely death. Intellectual Merit: The proposed work will contribute to a better understanding of declarative models of time and interaction. Broader Impacts: The proposed work will further broaden the applicability of FRP, improving implementation making FRP more suitable for compute-intensive applications, such as interactive 3D graphics and real-time audio processing. As the part of that grant we also developed and published ConfigC (acknowledgments to the grant were given). Publications: M. Santolucito, E. Zhai, R. Piskac: "Probabilistic Automated Language Learning for Configuration Files" (July 2016), CAV 2016, Pages 80-87, DOI: 10.1007/978-3-319-41540-6_5.

Bibliography

1 Facebook, Tinder, Instagram suffer widespread issues. `http://mashable.com/2015/01/27/facebook-tinder-instagram-issues/`.

2 Fine-grained value correlation error. `http://serverfault.com/questions/628414/my-cnf-configuration-in-mysql-5-6-x`.

3 Misconfiguration dataset. `https://github.com/tianyin/configuration_datasets`.

4 Stack Overflow. `http://stackoverflow.com/`.

5 Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. Opentuner: An extensible framework for program autotuning. In *International Conference on Parallel Architectures and Compilation Techniques*, Edmonton, Canada, August 2014.

6 Mona Attariyan, Michael Chow, and Jason Flinn. X-ray: Automating root-cause diagnosis of performance anomalies in production software. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, October 2012.

7 Mona Attariyan and Jason Flinn. Automating configuration troubleshooting with dynamic information flow analysis. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, October 2010.

8 Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 171–177, 2011.

9 Zaidman A. Beller M, Gousios G. Oops, my tests broke the build: An analysis of travis ci builds with github. PREPRINT, 2016.

10 François Bobot, Jean-Christophe Filliâtre, Claude Marché, and Andrei Paskevich. Let's verify this with why3. *STTT*, 17(6):709–727, 2015.

11 Xu Chen, Yun Mao, Zhuoqing Morley Mao, and Jacobus E. van der Merwe. Declarative configuration management for complex and dynamic networks. In *ACM CoNEXT (CoNEXT)*, November 2010.

12 Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 337–340, 2008.

13 William Enck, Patrick Drew McDaniel, Subhabrata Sen, Panagiotis Sebos, Sylke Spoerel, Albert G. Greenberg, Sanjay G. Rao, and William Aiello. Configuration management at massive scale: System design and experience. In *USENIX Annual Technical Conference (USENIX ATC)*, June 2007.

14 Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Unsupervised learning. In *The elements of statistical learning*, pages 485–585. Springer, 2009.

15 Peng Huang, William J. Bolosky, Abhishek Singh, and Yuanyuan Zhou. Confvalley: A systematic configuration validation framework for cloud services. In *10th European Conference on Computer Systems (EuroSys)*, April 2015.

16 John Launchbury and Simon L. Peyton Jones. Lazy functional state threads. In *Programming Language Design and Implementation (PLDI)*, pages 24–35. ACM Press, 1993.

17 K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. In *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16*, pages 348–370, 2010.

18 Boon Thau Loo, Joseph M. Hellerstein, Ion Stoica, and Raghu Ramakrishnan. Declarative routing: Extensible routing with declarative queries. In *ACM SIGCOMM (SIGCOMM)*, August 2005.

19 Tom M Mitchell. Version spaces: A candidate elimination approach to rule learning. In Proceedings of the 5th international joint conference on Artificial intelligence-Volume 1, pages 305-310. Morgan Kaufmann Publishers Inc., 1977.

20 Ruzica Piskac, Thomas Wies, and Damien Zufferey. Grasshopper - complete heap verification with mixed specifications. In *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014*, pages 124–139, 2014.

21 Veselin Raychev, Pavol Bielik, Martin T. Vechev, and Andreas Krause. Learning programs from noisy data. In *43rd ACM SIGPLAN-SIGACT (POPL) Symposium on Principles of Programming Languages*, January 2016.

22 Veselin Raychev, Martin T. Vechev, and Andreas Krause. Predicting program properties from "big code". In *42nd ACM SIGPLAN-SIGACT (POPL) Symposium on Principles of Programming Languages*, January 2015.

23 Mark Santolucito, Ennan Zhai, and Ruzica Piskac. Probabilistic automated language learning for configuration files. In *28th Computer Aided Verification (CAV)*, July 2016.

24 Ya-Yunn Su, Mona Attariyan, and Jason Flinn. AutoBash: Improving configuration management with operating systems. In *21st ACM Symposium on Operating Systems Principles (SOSP)*, October 2007.

25 Helen J. Wang, John C. Platt, Yu Chen, Ruyun Zhang, and Yi-Min Wang. Automatic misconfiguration troubleshooting with PeerPressure. In *6th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, December 2004.

26 Andrew Whitaker, Richard S. Cox, and Steven D. Gribble. Configuration debugging as search: Finding the needle in the haystack. In *6th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, December 2004

27 Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. Key, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software. In *10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, August 2015.

28 Tianyin Xu, Xinxin Jin, Peng Huang, Yuanyuan Zhou, Shan Lu, Long Jin, and Shankar Pasupathy. Early detection of configuration errors to reduce failure damage. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, November 2016.

29 Tianyin Xu, Jiaqi Zhang, Peng Huang, Jing Zheng, Tianwei Sheng, Ding Yuan, Yuanyuan Zhou, and Shankar Pasupathy. Do not blame users for misconfigurations. In *24th ACM Symposium on Operating Systems Principles (SOSP)*, November 2013.

30 Tianyin Xu and Yuanyuan Zhou. Systems approaches to tackling configuration errors: A survey. *ACM Comput. Surv.*, 47(4):70, 2015.

31 Zuoning Yin, Xiao Ma, Jing Zheng, Yuanyuan Zhou, Lakshmi N. Bairavasundaram, and Shankar Pasupathy. An empirical study on configuration errors in commercial and open source systems. In *23rd ACM Symposium on Operating Systems Principles (SOSP)*, October 2011.

32 Ding Yuan, Yinglian Xie, Rina Panigrahy, Junfeng Yang, Chad Verbowski, and Arunvijay Kumar. Context-based online configuration-error detection. In *USENIX Annual Technical Conference (USENIX ATC)*, June 2011.

33 Jiaqi Zhang, Lakshminarayanan Renganarayana, Xiaolan Zhang, Niyu Ge, Vasanth Bala, Tianyin Xu, and Yuanyuan Zhou. Encore: Exploiting system environment and correlation information for misconfiguration detection. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2014.

Ruzica Piskac

Professional Preparation

University of Zagreb, Croatia	Mathematics	B.Sc.	2000
Max Planck Institute for Computer Science, Germany	Computer Science	M.Sc.	2005
EPFL, Switzerland	Computer Science	Ph.D.	2011

Appointments

September 2013 –	Asst. Prof. of Comp. Sci., Yale
January 2012 – August 2013	Faculty (tenure track), Max Planck Institute for Software Systems, Germany

Five Products Most Closely Related to the Proposed Project

1. M. Santolucito, E. Zhai, R. Piskac. "Probabilistic Automated Language Learning for Configuration Files." *Proceedings of the 28th International Conference on Computer Aided Verification (CAV 2016)*, p. 80-87
2. A. Reinking, R. Piskac. "A Type-Directed Approach to Program Repair." *Proceedings of the 27th International Conference on Computer Aided Verification (CAV 2015)*, p. 511-517
3. T. Gvero, V. Kuncak, I. Kuraj, R. Piskac. "Complete Completion using Types and Weights." *Proceedings of the ACM SIGPLAN 2013 Conference on Programming Language Design and Implementation (PLDI)*, p. 27-38.
4. V. Kuncak, M. Mayer, R. Piskac, P. Suter. "Software Synthesis Procedures." *Communications of the ACM (55)*, Volume 55, Number 2, p. 103-111, DOI: 10.1145/2076450.2076472 February 2012.
5. R. Piskac, T. Wies, D. Zufferey. "Automating Separation Logic with Trees and Data." *Proceedings of the 26th International Conference on Computer Aided Verification (CAV 2014)*, Springer, LNCS Volume 8559, p. 711-728.

Five Other Significant Products

1. R. Piskac, T. Wies, D. Zufferey. "GRASShopper - Complete Heap Verification with Mixed Specifications." *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2014)*, p. 124- 139.
2. S. Gulwani, M. Mayer, F. Niksic, R. Piskac. "StriSynth: Synthesis for Live Programming." *ICSE 2015 (The 37th International Conference on Computer Aided Verification (CAV 2014) Software Engineering), Demonstrations Track*
3. R. Piskac, V. Kuncak. "Linear Arithmetic with Stars." *Proceedings of the 20th International Conference on Computer Aided Verification (CAV 2008)*, Springer, LNCS 5123, p. 268-280.
4. V. Kuncak, M. Mayer, R. Piskac, P. Suter. "Complete Functional Synthesis." *Proceedings of the ACM SIGPLAN 2010 Conference on Programming Language Design and Implementation (PLDI)*, p. 316-329.
5. R. Piskac, T. Wies, D. Zufferey. "Automating Separation Logic Using SMT." *Proceedings of the 25th International Conference on Computer Aided Verification (CAV 2013)*, p. 773-789.

Synergistic Activities (Examples)

1. Program committee member (recent and partial): ESOP 2017, CAV 2106, PLDI 2015, VMCAI 2014 & 2015, FMCAD 2013 & 2014 & 2015, FTfJP 2015, GandALF 2015, TACAS 2015, CADE 2015, ESEC/FSE 2013 & 2014 Tool Demos track, CSL 2013, INFINITY 2012, PAAR-2012, External Review Committee of PLDI 2014.

2. Referee for (partial list): *Handbook of Model Checking*, *Journal of Symbolic Computation*, *STTT Journal*.

3. Publicity Chair for POPL 2015 and 2016, VMCAI 2014, 2015 and 2016

4. Organizer of mentoring workshops for PL/Verification students at POPL 2015 and CAV 2015 and 2016.

5. Organizer of the SYNT 2016 workshop, SMT 2016 workshop and SMT/SAT/AR summer school 2016.

6. General and PC Chair for the FMCAD 2016 conference

SUMMARY PROPOSAL BUDGET

YEAR 1

ORGANIZATION Yale University				FOR NSF USE ONLY PROPOSAL NO.	DURATION (months) Proposed / Granted
PRINCIPAL INVESTIGATOR / PROJECT DIRECTOR Ruzica Piskac				AWARD NO.	

A. SENIOR PERSONNEL: PI/PD, Co-PI's, Faculty and Other Senior Associates (List each separately with title, A.7. show number in brackets)	NSF Funded Person-months CAL	ACAD	SUMR	Funds Requested By proposer	Funds granted by NSF (if different)
1. Ruzica Piskac - PI	0.00	0.00	0.75	9,390	
2.					
3.					
4.					
5.					
6. (0) OTHERS (LIST INDIVIDUALLY ON BUDGET JUSTIFICATION PAGE)	0.00	0.00	0.00	0	
7. (1) TOTAL SENIOR PERSONNEL (1 - 6)	0.00	0.00	0.75	9,390	
B. OTHER PERSONNEL (SHOW NUMBERS IN BRACKETS)					
1. (1) POST DOCTORAL SCHOLARS	12.00	0.00	0.00	48,900	
2. (0) OTHER PROFESSIONALS (TECHNICIAN, PROGRAMMER, ETC.)	0.00	0.00	0.00	0	
3. (1) GRADUATE STUDENTS				8,536	
4. (0) UNDERGRADUATE STUDENTS				0	
5. (0) SECRETARIAL - CLERICAL (IF CHARGED DIRECTLY)				0	
6. (0) OTHER				0	
TOTAL SALARIES AND WAGES (A + B)				66,826	
C. FRINGE BENEFITS (IF CHARGED AS DIRECT COSTS)				18,128	
TOTAL SALARIES, WAGES AND FRINGE BENEFITS (A + B + C)				84,954	
D. EQUIPMENT (LIST ITEM AND DOLLAR AMOUNT FOR EACH ITEM EXCEEDING $5,000.)					
TOTAL EQUIPMENT				0	
E. TRAVEL 1. DOMESTIC (INCL. U.S. POSSESSIONS)				5,000	
2. FOREIGN				5,000	
F. PARTICIPANT SUPPORT COSTS 1. STIPENDS $ 0 2. TRAVEL 0 3. SUBSISTENCE 0 4. OTHER 0					
TOTAL NUMBER OF PARTICIPANTS (0) TOTAL PARTICIPANT COSTS				0	
G. OTHER DIRECT COSTS					
1. MATERIALS AND SUPPLIES				2,500	
2. PUBLICATION COSTS/DOCUMENTATION/DISSEMINATION				0	
3. CONSULTANT SERVICES				0	
4. COMPUTER SERVICES				0	
5. SUBAWARDS				0	
6. OTHER				0	
TOTAL OTHER DIRECT COSTS				2,500	
H. TOTAL DIRECT COSTS (A THROUGH G)				97,454	
I. INDIRECT COSTS (F&A)(SPECIFY RATE AND BASE) MTDC On Campus (Rate: 67.5000, Base: 97455)					
TOTAL INDIRECT COSTS (F&A)				65,782	
J. TOTAL DIRECT AND INDIRECT COSTS (H + I)				163,236	
K. SMALL BUSINESS FEE				0	
L. AMOUNT OF THIS REQUEST (J) OR (J MINUS K)				163,236	

M. COST SHARING PROPOSED LEVEL $ 0	AGREED LEVEL IF DIFFERENT $		
PI/PD NAME Ruzica Piskac	FOR NSF USE ONLY INDIRECT COST RATE VERIFICATION		
ORG. REP. NAME*	Date Checked	Date Of Rate Sheet	Initials - ORG

*ELECTRONIC SIGNATURES REQUIRED FOR REVISED BUDGET

SUMMARY PROPOSAL BUDGET

YEAR 2

ORGANIZATION Yale University				FOR NSF USE ONLY PROPOSAL NO.	DURATION (months) Proposed / Granted
PRINCIPAL INVESTIGATOR / PROJECT DIRECTOR Ruzica Piskac				AWARD NO.	

A. SENIOR PERSONNEL: PI/PD, Co-PI's, Faculty and Other Senior Associates (List each separately with title, A.7. show number in brackets)	NSF Funded Person-months CAL	ACAD	SUMR	Funds Requested By proposer	Funds granted by NSF (if different)
1. Ruzica Piskac - PI	0.00	0.00	0.75	9,672	
2.					
3.					
4.					
5.					
6. (0) OTHERS (LIST INDIVIDUALLY ON BUDGET JUSTIFICATION PAGE)	0.00	0.00	0.00	0	
7. (1) TOTAL SENIOR PERSONNEL (1 - 6)	0.00	0.00	0.75	9,672	
B. OTHER PERSONNEL (SHOW NUMBERS IN BRACKETS)					
1. (1) POST DOCTORAL SCHOLARS	12.00	0.00	0.00	50,367	
2. (0) OTHER PROFESSIONALS (TECHNICIAN, PROGRAMMER, ETC.)	0.00	0.00	0.00	0	
3. (1) GRADUATE STUDENTS				8,792	
4. (0) UNDERGRADUATE STUDENTS				0	
5. (0) SECRETARIAL - CLERICAL (IF CHARGED DIRECTLY)				0	
6. (0) OTHER				0	
TOTAL SALARIES AND WAGES (A + B)				68,831	
C. FRINGE BENEFITS (IF CHARGED AS DIRECT COSTS)				18,672	
TOTAL SALARIES, WAGES AND FRINGE BENEFITS (A + B + C)				87,503	
D. EQUIPMENT (LIST ITEM AND DOLLAR AMOUNT FOR EACH ITEM EXCEEDING $5,000.)					
TOTAL EQUIPMENT				0	
E. TRAVEL 1. DOMESTIC (INCL. U.S. POSSESSIONS)				5,000	
2. FOREIGN				5,000	
F. PARTICIPANT SUPPORT COSTS 1. STIPENDS $ 0 2. TRAVEL 0 3. SUBSISTENCE 0 4. OTHER 0					
TOTAL NUMBER OF PARTICIPANTS (0) TOTAL PARTICIPANT COSTS				0	
G. OTHER DIRECT COSTS					
1. MATERIALS AND SUPPLIES				1,500	
2. PUBLICATION COSTS/DOCUMENTATION/DISSEMINATION				0	
3. CONSULTANT SERVICES				0	
4. COMPUTER SERVICES				0	
5. SUBAWARDS				0	
6. OTHER				0	
TOTAL OTHER DIRECT COSTS				1,500	
H. TOTAL DIRECT COSTS (A THROUGH G)				99,003	
I. INDIRECT COSTS (F&A)(SPECIFY RATE AND BASE) MTDC On Campus (Rate: 67.5000, Base: 99004)					
TOTAL INDIRECT COSTS (F&A)				66,828	
J. TOTAL DIRECT AND INDIRECT COSTS (H + I)				165,831	
K. SMALL BUSINESS FEE				0	
L. AMOUNT OF THIS REQUEST (J) OR (J MINUS K)				165,831	

M. COST SHARING PROPOSED LEVEL $ 0	AGREED LEVEL IF DIFFERENT $		
PI/PD NAME Ruzica Piskac	FOR NSF USE ONLY INDIRECT COST RATE VERIFICATION		
ORG. REP. NAME*	Date Checked	Date Of Rate Sheet	Initials - ORG

*ELECTRONIC SIGNATURES REQUIRED FOR REVISED BUDGET

SUMMARY PROPOSAL BUDGET

YEAR 3

ORGANIZATION **Yale University**				FOR NSF USE ONLY PROPOSAL NO.	DURATION (months) Proposed	Granted
PRINCIPAL INVESTIGATOR / PROJECT DIRECTOR **Ruzica Piskac**				AWARD NO.		

A. SENIOR PERSONNEL: PI/PD, Co-PI's, Faculty and Other Senior Associates (List each separately with title, A.7. show number in brackets)	NSF Funded Person-months CAL	ACAD	SUMR	Funds Requested By proposer	Funds granted by NSF (if different)
1. **Ruzica Piskac - PI**	0.00	0.00	0.75	**9,962**	
2.					
3.					
4.					
5.					
6. (**0**) OTHERS (LIST INDIVIDUALLY ON BUDGET JUSTIFICATION PAGE)	0.00	0.00	0.00	**0**	
7. (**1**) TOTAL SENIOR PERSONNEL (1 - 6)	0.00	0.00	0.75	**9,962**	
B. OTHER PERSONNEL (SHOW NUMBERS IN BRACKETS)					
1. (**1**) POST DOCTORAL SCHOLARS	12.00	0.00	0.00	**51,878**	
2. (**0**) OTHER PROFESSIONALS (TECHNICIAN, PROGRAMMER, ETC.)	0.00	0.00	0.00	**0**	
3. (**1**) GRADUATE STUDENTS				**9,056**	
4. (**0**) UNDERGRADUATE STUDENTS				**0**	
5. (**0**) SECRETARIAL - CLERICAL (IF CHARGED DIRECTLY)				**0**	
6. (**0**) OTHER				**0**	
TOTAL SALARIES AND WAGES (A + B)				**70,896**	
C. FRINGE BENEFITS (IF CHARGED AS DIRECT COSTS)				**19,232**	
TOTAL SALARIES, WAGES AND FRINGE BENEFITS (A + B + C)				**90,128**	
D. EQUIPMENT (LIST ITEM AND DOLLAR AMOUNT FOR EACH ITEM EXCEEDING $5,000.)					
TOTAL EQUIPMENT				**0**	
E. TRAVEL 1. DOMESTIC (INCL. U.S. POSSESSIONS)				**5,000**	
2. FOREIGN				**5,000**	
F. PARTICIPANT SUPPORT COSTS 1. STIPENDS $ **0** 2. TRAVEL **0** 3. SUBSISTENCE **0** 4. OTHER **0**					
TOTAL NUMBER OF PARTICIPANTS (**0**) TOTAL PARTICIPANT COSTS				**0**	
G. OTHER DIRECT COSTS					
1. MATERIALS AND SUPPLIES				**1,500**	
2. PUBLICATION COSTS/DOCUMENTATION/DISSEMINATION				**0**	
3. CONSULTANT SERVICES				**0**	
4. COMPUTER SERVICES				**0**	
5. SUBAWARDS				**0**	
6. OTHER				**0**	
TOTAL OTHER DIRECT COSTS				**1,500**	
H. TOTAL DIRECT COSTS (A THROUGH G)				**101,628**	
I. INDIRECT COSTS (F&A)(SPECIFY RATE AND BASE) **MTDC On Campus (Rate: 67.5000, Base: 101629)**					
TOTAL INDIRECT COSTS (F&A)				**68,600**	
J. TOTAL DIRECT AND INDIRECT COSTS (H + I)				**170,228**	
K. SMALL BUSINESS FEE				**0**	
L. AMOUNT OF THIS REQUEST (J) OR (J MINUS K)				**170,228**	

M. COST SHARING PROPOSED LEVEL $ **0**	AGREED LEVEL IF DIFFERENT $		
PI/PD NAME **Ruzica Piskac**	FOR NSF USE ONLY INDIRECT COST RATE VERIFICATION		
ORG. REP. NAME*	Date Checked	Date Of Rate Sheet	Initials - ORG

*ELECTRONIC SIGNATURES REQUIRED FOR REVISED BUDGET

SUMMARY PROPOSAL BUDGET

Cumulative

ORGANIZATION: Yale University			FOR NSF USE ONLY: PROPOSAL NO.	DURATION (months) Proposed	Granted
PRINCIPAL INVESTIGATOR / PROJECT DIRECTOR: Ruzica Piskac			AWARD NO.		

A. SENIOR PERSONNEL: PI/PD, Co-PI's, Faculty and Other Senior Associates (List each separately with title, A.7. show number in brackets)	NSF Funded Person-months CAL	ACAD	SUMR	Funds Requested By proposer	Funds granted by NSF (if different)
1. Ruzica Piskac - PI	0.00	0.00	2.25	29,024	
2.					
3.					
4.					
5.					
6. ( ) OTHERS (LIST INDIVIDUALLY ON BUDGET JUSTIFICATION PAGE)	0.00	0.00	0.00	0	
7. (1) TOTAL SENIOR PERSONNEL (1 - 6)	0.00	0.00	2.25	29,024	
B. OTHER PERSONNEL (SHOW NUMBERS IN BRACKETS)					
1. (3) POST DOCTORAL SCHOLARS	36.00	0.00	0.00	151,145	
2. (0) OTHER PROFESSIONALS (TECHNICIAN, PROGRAMMER, ETC.)	0.00	0.00	0.00	0	
3. (3) GRADUATE STUDENTS				26,384	
4. (0) UNDERGRADUATE STUDENTS				0	
5. (0) SECRETARIAL - CLERICAL (IF CHARGED DIRECTLY)				0	
6. (0) OTHER				0	
TOTAL SALARIES AND WAGES (A + B)				206,553	
C. FRINGE BENEFITS (IF CHARGED AS DIRECT COSTS)				56,032	
TOTAL SALARIES, WAGES AND FRINGE BENEFITS (A + B + C)				262,585	
D. EQUIPMENT (LIST ITEM AND DOLLAR AMOUNT FOR EACH ITEM EXCEEDING $5,000.)					
TOTAL EQUIPMENT				0	
E. TRAVEL 1. DOMESTIC (INCL. U.S. POSSESSIONS)				15,000	
2. FOREIGN				15,000	
F. PARTICIPANT SUPPORT COSTS					
1. STIPENDS $ 0					
2. TRAVEL 0					
3. SUBSISTENCE 0					
4. OTHER 0					
TOTAL NUMBER OF PARTICIPANTS (0) TOTAL PARTICIPANT COSTS				0	
G. OTHER DIRECT COSTS					
1. MATERIALS AND SUPPLIES				5,500	
2. PUBLICATION COSTS/DOCUMENTATION/DISSEMINATION				0	
3. CONSULTANT SERVICES				0	
4. COMPUTER SERVICES				0	
5. SUBAWARDS				0	
6. OTHER				0	
TOTAL OTHER DIRECT COSTS				5,500	
H. TOTAL DIRECT COSTS (A THROUGH G)				298,085	
I. INDIRECT COSTS (F&A)(SPECIFY RATE AND BASE)					
TOTAL INDIRECT COSTS (F&A)				201,210	
J. TOTAL DIRECT AND INDIRECT COSTS (H + I)				499,295	
K. SMALL BUSINESS FEE				0	
L. AMOUNT OF THIS REQUEST (J) OR (J MINUS K)				499,295	

M. COST SHARING PROPOSED LEVEL $ 0	AGREED LEVEL IF DIFFERENT $		
PI/PD NAME: Ruzica Piskac	FOR NSF USE ONLY: INDIRECT COST RATE VERIFICATION		
ORG. REP. NAME*	Date Checked	Date Of Rate Sheet	Initials - ORG

C *ELECTRONIC SIGNATURES REQUIRED FOR REVISED BUDGET

BUDGET JUSTIFICATION

Personnel:
PI: Prof. Ruzica Piskac, PhD: (0.75 summer) The PI will be responsible for the overall administration and direction of the project. She will oversee research strategy decisions, establish and critique experimental design, and aid in the interpretation of data, mentoring the student on the project, and reporting of results.

Postdoctoral Associate (TBN): (12.0 calendar) Funds are budgeted for a postdoc during each year of the project. Since the proposed project is to apply formal methods in a topic that is traditionally considered to be a part of system research, the PDA should have a background in systems research to contextualize our methods. The PDA will be responsible for the initial design and the implementation of the proposed framework.

Graduate Research Assistant (TBN): (3.0 calendar) funds for a graduate student is requested. The student will assist in research as described in the project description and focus on developing an intermediary language for describing configuration files and developing algorithms to reason about them.

Fringes Benefits: Fringes for the PI are calculated at 31.1% per Yale's current rates.

Travel:
The travel budget includes two domestic and one foreign trip in each year for project personnel to attend conferences and workshops to present results. Estimated costs to cover transportation, lodging, subsistence and conference registration are based on past trips of similar nature. Leading conferences in the field relevant to the proposed research include examples below which alternate between locations domestically and internationally.

- ACM International Conference on Functional Programming (ICFP): ICFP 2017 will be held in Oxford, UK; in 2016 it was held in Nara, Japan and ICFP 2018 will be held in St. Louis, MO, USA.
- ACM Principles of Programming Languages (POPL): POPL 2017 will be held in Paris, France; POPL 2016 was held in St. Petersburg, FL, USA; POPL 2015 was held in Mumbai, India.
- ACM Programming Language Design and Implementation (PLDI): PLDI 2017 will be held in Barcelona, Spain; PLDI 2016 was held in Santa Barbara, CA, USA; PLDI 2018 is still TBD.
- International Conference on Computer Aided Verification (CAV): CAV 2017 will be held in Heidelberg Germany, CAV 2018 will be held in Melbourne, Australia; CAV 2016 was held in Toronto, Canada; and, CAV 2015 was held in San Francisco, CA, USA

Domestic Conference or Workshop: (1pp)		Foreign Conference or Workshop: (1-2 pp)	
Transportation (air/ground)	420	Transportation (r/t, air/ground)	1,500
Accommodations (3 nights)	660	Accommodations (5 nights)	1,712
Meals (4 days)	236	Meals (5.5 days)	838
Registration	350	Registration	950
Total:	***~1,666***	***Total:***	***~5,000***

Materials and Supplies:
We propose to develop an automated framework, to teach both the foundations of verification and their application in systems research, introducing in this way formal methods to the next generation

of engineers. Our tools should also be used by researchers. Necessary for the extensive data analysis to be performed, funds are requested to cover the purchase of computer accessories, software, machine maintenance and upgrades needed to complete the proposed research.

Indirect:
Indirect costs are calculated at Yale's federally negotiated rate of 67.5% of modified total direct costs. DHHS agreement dated 02/23/16.

Current and Pending Support

(See GPG Section II.C.2.h for guidance on information to include on this form.)

The following information should be provided for each investigator and other senior personnel. Failure to provide this information may delay consideration of this proposal.

Investigator: Ruzica Piskac

Other agencies (including NSF) to which this proposal has been/will be submitted.

Support: [✓ Current] ☐ Pending ☐ Submission Planned in Near Future ☐ *Transfer of Support

Project/Proposal Title: CAREER: Synthesis in a Live Programming Environment

Source of Support: NSF

Total Award Amount: $ 463,273 Total Award Period Covered: 01/01/16 - 12/31/20

Location of Project: Yale University

Person-Months Per Year Committed to the Project. Cal:0.00 Acad: 0.00 Sumr: 1.00

Support: [✓ Current] ☐ Pending ☐ Submission Planned in Near Future ☐ *Transfer of Support

Project/Proposal Title: Student Travel Support for SAT/SMT/AR Summer School at IJCAR 2016

Source of Support: NSF

Total Award Amount: $ 30,000 Total Award Period Covered: 05/01/16 - 04/30/17

Location of Project: Yale University

Person-Months Per Year Committed to the Project. Cal:0.00 Acad: 0.00 Sumr:

Support: [✓ Current] ☐ Pending ☐ Submission Planned in Near Future ☐ *Transfer of Support

Project/Proposal Title: SHF: Medium: Collaborative Research: FRP for Real

Source of Support: NSF

Total Award Amount: $ 866,000 Total Award Period Covered: 10/01/13 - 09/30/17

Location of Project: Yale University

Person-Months Per Year Committed to the Project. Cal:0.00 Acad: 0.00 Sumr: 1.00

Support: [✓ Current] ☐ Pending ☐ Submission Planned in Near Future ☐ *Transfer of Support

Project/Proposal Title: Student Travel Support for VMCAI 2015

Source of Support: NSF

Total Award Amount: $ 19,992 Total Award Period Covered: 12/01/14 - 11/30/16

Location of Project: Yale University

Person-Months Per Year Committed to the Project. Cal:0.00 Acad: 0.00 Sumr: 0.00

Support: ☐ Current [✓ Pending] ☐ Submission Planned in Near Future ☐ *Transfer of Support

Project/Proposal Title: SHF: Small: ConfigV: Automated Verification of Configuration Files (this proposal)

Source of Support: NSF

Total Award Amount: $499,295 Total Award Period Covered: 06/01/17 - 05/31/20

Location of Project: Yale University

Person-Months Per Year Committed to the Project. Cal: Acad: 0.00 Sumr: 0.75

---

*If this project has previously been funded by another agency, please list and furnish information for immediately preceding funding period.

Facilities: Yale Department of Computer Science

Yale University and Department of Computer Science provide resources that are relevant to the proposed research. The faculty, researchers, and students in the Department of Computer Science have access to a wide variety of ever-changing state-of-the-art computing resources, ranging from laptops, conventional PCs and scientific workstations to high-powered compute-servers and workstation clusters used as parallel computers. The PI's research group has access to the high-performance Bulldog clusters, which have over 1200 nodes, over 13,000 cores, and over 2.5 petabytes of high-performance storage.

Yale students in computer science, both graduate and undergraduate, have liberal access to all of these facilities. In this way students play a vital role in contributing to our understanding of theoretical and experimental issues in computer science.

The Departments computing resources are professionally managed by Faculty Support (FASIT), a unit of the Yale office of Information and Technology Services (ITS). FASIT staff follow policy set by a faculty oversight committee in providing first-class responsive service to all departmental users.

Further server hosting and administration is provided by Yale University available in a wide variety of reliable, secure, and cost-effective solutions. Yale ITS offers cloud-based, virtual, and data center solutions. This includes (i) Hosting for computing and storage systems in a secure, scalable, and environmentally controlled space in a Yale Data Center with redundant facilities and resilient technologies. (ii) Virtual server (Sprout at Yale) is a client managed, easy and affordable way to get your server in the Yale cloud. It offers great flexibility and users have full administrator privileges to customize the server. (iii) The ITS private cloud is based on redundant, highly reliable infrastructure with replicated storage and load and performance management.

DATA MANAGEMENT PLAN

Data Generated

We expect to generate the following artifacts:

1. Source code
2. Benchmarks
3. Project web pages
4. Published papers and technical reports

Availability of Data

We will make all data available from our website. Source code will be released under a BSD-like license, making it free to use for any purpose. When reporting experimental results, we will provide a companion web page with enough information to reproduce the results precisely.

Data Management Plan

The proposed project will produce data in the form of source code and measurement results. The source code developed in this project will consist chiefly of the systems described in the project narrative, as well as application benchmarks for evaluating the developed systems. The source code will be in various languages: Haskell, Javascript, Java, C++, C, Perl, Python, and custom languages that the PIs develop and document. The measurement results will be text files consisting of experimental logs, gathered by benchmarking logic inside the developed systems.

The PI will continue to release their source code and experimental configurations online with nonrestrictive licenses, accompanied by step-by-step HOWTOs; as a result, other researchers will be able to download this code, compile it, run it on the posted configurations, and thereby reproduce published results.

The PI will continue to manage their source code with git repositories that are backed up. The data will be preserved for at least three years beyond the award period, as required by NSF guidelines.

This project will not involve the acquisition of either animal or human subjects data.

Data Reproducibility

Many major conferences in the field of programming languages and verification (PLDI, POPL, CAV) have an additional committee, Artifact Evaluation Committee, that checks and verifies correctness of all reported results. The PI is actively submitting all her results to be formally verified, and whenever possible she has obtained the AEC badge (a stamp that says: “Artifact Evaluated: Consistent * Complete * Well Documented * Easy to Reuse”). This guarantees that anyone who downloads and runs our open source release will be able to generate the same data that we used in our papers. The PI will continue with that practice.

<u>POSTDOCTORAL RESEARCHER MENTORING PLAN</u>

The professional development of the postdoctoral researcher supported by this project will be enhanced through a program of structured mentoring activities. The goal of the mentoring program will be to provide the skills, knowledge and experience to prepare the postdoctoral researcher to excel in his/her career path. To accomplish this goal, the mentoring plan will follow the guidance of the National Academies of Science and Engineering on how to enhance the postdoctoral experience by providing a structured mentoring plan, career planning assistance, and opportunities to learn a number of career skills such as writing grant proposals, teaching students, writing articles for publication and communication skills. Specific elements of the mentoring plan will include:

1) Participation in seminars on teaching and learning, as well as access to a teaching mentoring program.
2) Individual consultations and practical experience on how to identify research funding opportunities and write competitive proposals. The Postdoctoral Researcher will have an opportunity to learn best practices in proposal preparation including identification of key research questions, definition of objectives, description of approach and rationale, and construction of a work plan, timeline, and budget.
3) Opportunities to network with visiting scholars who are leaders in our field when they participate in the schools various visiting speaker series.
4) Travel to at least one conference each year, with the goal that the postdoctoral fellow present a poster or paper at the conference.
5) Publications and presentations are expected to result from the work supported by the grant. The postdoctoral researcher will receive guidance and training in the preparation of manuscripts for scientific journals and presentations at conferences;
6) Participation in, and management duties of, the PIs weekly research group meetings, in which members will be expected to present their research regularly, and feedback and coaching will be given to help all members to develop their communication and presentation skills.
7) Instruction in professional practices will be provided on a regular basis in the context of the research work and will include fundamentals of the scientific method and other standards of professional practice. In addition, the Postdoctoral Researcher will be encouraged to affiliate with one or more professional societies in his/her chosen field.
8) Technology Transfer activities will be covered when applicable, as the postdoctoral researcher will be given an opportunity to become familiar with the preparation of invention disclosure applications.

Success of this mentoring plan will be assessed by tracking the progress of the postdoctoral fellow through her/his Individual Development Plan, periodic discussions of the postdoctoral fellow to assess satisfaction with the mentoring program, and tracking of the postdoctoral fellows progress toward his/her career goals after finishing the postdoctoral position