CSV WORKSHOP

On May, 20th 2022 Ca’ Foscari University will award prof. Patrick Cousot with a PhD in Computer Science Honoris Causa. The ceremony will take place at 11am in the Aula Mario Baratto. In the afternoon, the workshop on “Challenges of Software Verification” will take place in the same room. The workshop is by invitation only, and it will consist of 20’ talks followed by a Q&A session. In addition, the week after (from May, 23rd to May, 27th) Patrick will give a course on “Principles of Abstract Interpretation”.

**Schedule**

<table>
<thead>
<tr>
<th>Time</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>11:00-</td>
<td>Patrick Cousot Honoris Causa PhD cerimony</td>
</tr>
<tr>
<td>13:00</td>
<td></td>
</tr>
</tbody>
</table>
13:00-14:00 Lunch

Application challenges (chair: Vincenzo Arceri)
14.00 – 14.15: Static Analysis for Data Scientists (Urban)
14.15 – 14.30: Scaling security at Meta with Abstract Interpretation (Logozzo)
14.30 – 14.45: Modular Multi-Language Static Analysis in LiSA (Ferrara)

14:00-15:00 Static analysis by abstract interpretation for multiple languages and multi-language programs (Miné)
15.00 – 15.15: Using Synthesis for Modular Verification (Piskac)
15.15 – 15.30: Floating-point round-off error analysis of safety-critical avionics software (Titolo)
15.30 – 15.45: “Fixing” the specification of widenings (Zaffanella)

16:00-16:30 Coffee Break

Theoretical challenges (chair: Martina Olliaro)
16.30 – 16.45 Completeness in abstract interpretation (Monniaux)
16.45 – 17.00 A Correctness/Incorrectness Program Logic based on Abstract Interpretation (Ranzato)
17.00 – 17.15 The Topdown Solver: an Exercise in A2I (Seidl)
16:30-17:15 Software verification: from safety to security (Jensen)
17.30 – 17.45 Modular Verification of Safe Rust Programs (Mueller)
17.45 – 18.00 Abstraction and Induction in Polynomial Probabilistic Programs (Kovacs)
18.00 – 18.15 String constraint solving: past, present, and future (Amadini)
18.15 – 18.30 Risk estimation in IoT systems (Bodei)

19:30 Social Dinner

**Organizing committee**

**Agostino Cortesi** (University of Venice), chair

**Pietro Ferrara** (University of Venice), program co-chair

**Vincenzo Arceri** (University of Parma), program co-chair

**Martina Olliaro** (University of Venice), local chair

**Invited talks (alphabetical order)**

**String constraint solving: past, present and future**

**Roberto Amadini, University of Bologna (Italy)**

String constraint solving refers to solving combinatorial problems involving constraints over string variables. String solving approaches have become popular over the past few years given the massive use of strings in different application domains like formal analysis, automated testing, database query processing, and cybersecurity. We
provide a short survey of different string constraint solving approaches, by highlighting the state-of-the-art and identifying future challenges.

**Risk estimation in IoT systems**

*Chiara Bodei, University of Pisa (Italy)*

In the era of the Internet of Things, it is essential to make sure that data collected by sensors and smart devices is reliable and that it is aggregated and transmitted securely to other nodes and servers. This can have an impact on critical decisions and actuations of IoT systems, with possibly serious consequences if linked to essential services.

We apply Control Flow Analysis to systems modelled in the IoT-LySa process calculus, in order to statically predict how data will flow through the system and form a kind of supply chain for subsequent aggregations and use.

We also provide graph-based modelling and risk analysis that captures what dependencies exist between collected and aggregated data and critical decisions.

**Modular Multi-Language Static Analysis in LiSA**

*Pietro Ferrara, University of Venice (Italy)*

One of the biggest challenges in static analysis is multi-language analysis. Modern programs are often split into different modules written in different languages, and analyzing them in isolation provides
partial/inaccurate results, while formalizing an analysis that treat the same construct differently depending on the semantics of the source language is challenging. The aim of this talk is to introduce LiSA, a library for static analysis that focuses on multi-language analysis. LiSA is also modular, meaning that each component of the analysis has no knowledge on what lies outside of its scope: this enables effortless plugging of different components in different analyses to obtain different levels of precision.

Software verification: from safety to security

Thomas Jensen, INRIA Rennes (France)

Abstract interpretation and program logics have been successful in verifying the safety of software. This has resulted in scalable techniques with firm theoretical foundations that can serve to certify software safety. This talk will address the question of how to extend such logics to obtain similar powerful tools for reasoning about security. Particular emphasis will be given on developing a logic for reasoning about secure flow of information.

Abstraction and Induction in Polynomial Probabilistic Programs

Laura Kovacs, TU Wien (Austria)

We describe a static analysis approach towards establishing safety and termination of probabilistic while-programs whose guards and expressions are
polynomial expressions over random variables and parametrised distributions.
We abstract loops through recurrence equations over loop variables and loop iterations, allowing us to combine methods from symbolic summation and statistics to further derive inductive loop invariants. Our loop invariants are valid properties over higher-order moments, such as expected values or variances, of program variables, synthesizing this way quantitative invariants of probabilistic program loops. We further extend our moments-based analysis to prove loop termination, by providing polynomial witnesses for (positive) almost sure termination of programs and automating termination proof rules using constraints on supermartingales.

**Scaling security at Meta with Abstract Interpretation**

Francesco Logozzo, Meta (U.S.A.)

Over 50% of the security vulnerabilities we found across Meta’s family of apps (Facebook, Instagram, WhatsApp, Messenger, Oculus…) are detected automatically using Abstract Interpretation-based tools. In the talk I will present the challenges we faced (accuracy, scale, usability, customization, inter-language analysis) and how we achieved that result.

We worked in conjunction with the Meta Product Security team to focus on the bugs that matter and to constantly refine the analysis results. We designed new abstract domains, implemented a modular,
compositionally, non-uniform, parallel, and distributed analysis so to analyze hundreds of millions of lines of code in less than one hour and flag security vulnerabilities at code review time, preventing security bugs to land in production code. We built a system that let us achieve inter-language analysis, and a generic filtering system based on breadcrumbs that enable security engineers to customize the signal-to-noise ratio. For instance, a security engineer was able to increase the signal-to-noise ratio of results from 20% to 70% for SQL injection, by simply adding a filter on integer breadcrumbs. I will conclude the talk by debunking some myths on modular/parallel/distributed analyses, e.g. that modular implies scalable and by sharing some directions on theoretical abstract interpretation that will have huge impact in practice.

**Static analysis by abstract interpretation for multiple languages and multi-language programs**

Antoine Miné, Sorbonne Université (France)

We present MOPSA, a platform for static analysis by abstract interpretation we are building. It strives to achieve a high degree of modularity and extensibility by considering value abstractions, iterators, and control-flow abstractions uniformly as domain modules, with common capabilities to extend the language syntax and collaborate through dynamic expression rewriting. The platform currently includes
a classic value analysis for C programs that checks for run-time errors, but also more original analyses such as type, value, and exception analyses for Python programs, a portability analysis for C programs, as well as work in progress on a value analysis for the Michelson smart contract language. Despite the significant variation in the languages analyzed and the properties inferred, these analyses share many common abstractions. Finally, we show the capabilities of MOPSA to handle several source languages seamlessly in a single analysis, which is applied to analyze Python programs that call C libraries. This is joint work with my PhD students and post-docs. MOPSA is available as open-source software at https://gitlab.com/mopsa/mopsa-analyzer

**Completeness in abstract interpretation**

David Monniaux, CNRS Verimag (France)

Most descriptions of abstract interpretation discuss the case where the domain has no infinite ascending chains, and the use of widening operators otherwise. Yet in some cases, it is possible to compute the least fixed point in the abstract domain (e.g. by some optimization or quantifier elimination methods); there is thus a complete algorithm for verification in that abstract domain. Would it be possible to have such an algorithm in the domain of convex polyhedra? The answer is negative if some transitions are nonlinear, but the question is open if transitions are linear. Finally (joint work with Julien Braine and Laure
Gonnord) we discuss a notion of relative completeness: an abstract reachability problem is transformed into another (still possibly undecidable) problem in a way that does not lose precision, that is, the second problem has a solution if and only if the first has one. We illustrate it with abstract domains of maps / arrays.

**Modular Verification of Safe Rust Programs**

Peter Mueller, ETH Zurich (Switzerland)

Rust’s type system ensures memory safety: well-typed Rust programs are guaranteed to not exhibit problems such as dangling pointers, data races, and unexpected side effects through aliased references. Ensuring correctness properties beyond memory safety, for instance, the guaranteed absence of assertion failures or more-general functional correctness, requires static program verification. For traditional system programming languages, formal verification is notoriously difficult and requires complex specifications and logics to reason about pointers, aliasing, and side effects on mutable state. This complexity is a major obstacle to the more-widespread verification of system software.

In this talk, we present a verification technique and tool that leverages Rust’s type system to greatly simplify the verification of system software written in Rust. Users can annotate Rust programs with assertions at the abstraction level of Rust
expressions; our technique combines them with information from the Rust compiler to verify modularly whether these specifications hold. Crucially, our proofs are constructed and checked automatically without exposing the underlying formal logic, allowing users to work exclusively at the level of abstraction of the programming language.

**Using Synthesis for Modular Verification**

Ruzica Piskac, Yale University (U.S.A.)

Modular verifiers, such as LiquidHaskell, Dafny, and ESC/Java, allow programmers to write and prove specifications. We use cookies on our website to give you the most relevant experience by remembering your preferences and repeat visits. By clicking “Accept”, you consent to the use of ALL the cookies. View our **Cookie Policy**.

verifying a function f, modular verifiers consider only the specification of a called function g, not the actual definition of g. Thus, a modular verifier may fail to prove a true specification of f if the specification of g is too weak. In this talk, we will present a technique, counterfactual symbolic execution, to aid in the debugging of modular verification failures. The approach uses symbolic execution to find concrete counterexamples, in the case of an actual inconsistency between a program and a specification; and abstract counterexamples, in the case that a function specification is too weak. Furthermore, a counterexample-guided inductive synthesis (CEGIS) loop based technique is introduced to fully automate the process of modular verification, by using found
counterexamples to automatically infer needed function specifications. The counterfactual symbolic execution and automated specification inference techniques are implemented in G2, and evaluated on existing LiquidHaskell errors and programs.

**A Correctness/Incorrectness Program Logic based on Abstract Interpretation**

Francesco Ranzato, University of Padova (Italy)

We introduce the notion of local completeness in abstract interpretation and define a program logic for proving both the correctness and incorrectness of some program specification. Our main contribution is the design of a proof system, parameterized by an abstract domain, that, for the first time, combines over- and under-approximations of program behaviours. It turns out that if the underlying abstraction is the trivial singleton abstraction then our program logic coincides with O’Hearn incorrectness logic.

**The Topdown Solver: an Exercise in A2I**

Helmut Seidl, TU Munich (Germany)

The topdown solver TD is a convenient local generic fixpoint engine which is at the heart of static analysis frameworks such as CIAO and Goblint. In this presentation, we show how Patrick’s idea of applying analysis to the analyzer itself, allows to derive advanced versions of TD from a simple recursive descent fixpoint algorithm. A run of that fixpoint
algorithm provides us with a trace whose dynamic analysis allows to identify widening/narrowing points as well as semantic dependencies between unknowns on the fly.

In this application, it is thus not only the sequence of iterates for individual unknowns which is taken into account, but the global behavior of the fixpoint algorithm itself. We also indicate how further online analyses can be introduced in order to selectively increase the precision of the analysis. Given some assertion in the program which the static analyzer failed to verify, we would like to pinpoint where in the run of the analyzer the precision was lost in order to selectively refine the analysis, e.g., by distinguishing calling-contexts or introducing more precise abstract domains before partially re-analyzing the system.

**Floating-point round-off error analysis of safety-critical avionics software**

Laura Titolo, NASA (U.S.A.)

The development of software that depends on floating-point computations is particularly challenging due to the presence of round-off errors in computer arithmetic. These errors accumulate during numerical computations and may significantly affect the evaluation of both arithmetic and Boolean expressions.

Reasoning on floating-point computations is particularly important for safety-critical software where a divergence between the ideal real number...
computation and its floating-point counterpart can lead to catastrophic consequences.

This talk provides an overview of the different formal methods techniques that have been implemented and integrated to analyze numerical properties of NASA libraries for avionics applications such as geofencing, detect-and-avoid, and aircraft positioning.

Employed techniques include abstract interpretation, theorem proving, SMT solvers, and numerical methods which have been successfully integrated to improve floating-point programs and to provide formal guarantees on their correctness.

**Static Analysis for Data Scientists**

Caterina Urban, INRIA Paris (France)

Big data analytics has revolutionized the world of software development in the past decade. Every day, data scientists write computer programs to clean, manipulate, and visualize data, in order to help us make data-driven decisions. As we rely more and more on data analytics software, we become increasingly vulnerable to programming or technical mistakes. Mistakes that do not cause software failures can have serious consequences, since they give no indication that something went wrong. A simple technical mistake made during data processing caused nearly 16,000 cases of Covid-19 between September 25th and October 2nd, 2020 to go unreported from official figures in the UK. As a
consequence, Public Health England was unable to send out the relevant contact-tracing alerts. Mistakes in safety-critical applications can be deadly. In this talk, I will present ongoing work to develop an abstract interpretation-based static analysis framework for data scientists. In particular, I will focus on an analysis that infers necessary conditions on the structure and values of the data read by a data analytics program. The analysis builds on a family of underlying abstract domains, extended to indirectly reason about the input data rather than simply reasoning about the program variables. The choice of these abstract domains is a parameter of the analysis. We describe various instances built from existing abstract domains. We then demonstrate the potential of the approach on a number of representative examples and discuss ongoing efforts to target data analytics using Jupyter notebooks.

“Fixing” the specification of widenings

Enea Zaffanella, University of Parma (Italy)

The development of parametric analysis tools based on Abstract Interpretation relies on a clean separation between a generic fixpoint approximation engine and its main parameter, the abstract domain: a safe integration requires that the engine uses each domain operator according to its specification. Widening operators are special, among other reasons, in that they lack a single, universally adopted specification.
In this talk, we review the specification and usage of widenings in a few open-source implementations of abstract domains and analysis tools, witnessing a mismatch that potentially affects correctness and thereby motivating the need to “fix” the widening specification. For the domain of convex polyhedra, we also show that the fixed specification can be implemented without a significant efficiency penalty.