Xin Zhang — Research Statement

Software is becoming increasingly pervasive and complex. The goal of my research is to help developers build software that is correct, reliable, secure, and efficient. Towards this end, my research interest lies in the intersection of programming languages and software engineering with an emphasis on program analysis, a technique that automatically reasons about program properties of interest.

Practical program analysis tools have demonstrated the ability to prove non-trivial properties of real-world software or find critical defects related to violations of these properties. Examples of such tools include Microsoft Static Driver Verifier, Astrée, Coverity, and Facebook Infer. Due to the undecidable nature of program analysis problems in general, however, these tools inevitably make approximations. These approximations, collectively called the *abstraction*, control the soundness, accuracy, and scalability of the tool. State-of-the-art program analysis tools rely on an expert designer to carefully choose an abstraction that they deem appropriate for all possible usage scenarios. However, due to the rich variety in analysis user expectations, individual properties of interest, and characteristics of the considered programs, there is no such one-size-fits-all abstraction. As a result, analysis tools often fail to meet the needs of individual usage scenarios, which greatly hinders their soundness, accuracy, and scalability.

My thesis research addresses this challenge by proposing **a user-centric approach to program analysis**. My key insight is that, instead of pursuing a one-size-fits-all abstraction when designing the analysis, we can on the fly tailor the abstraction to the needs of individual usage scenarios. Such needs concern the feedback from the analysis users, the assertions of interest, and the characteristics of the subject programs.

I addressed two central technical challenges in order to enable a user-centric approach to program analysis:

1. *How can we adapt an existing analysis that has a fixed abstraction and is therefore rigid to a given usage scenario?*
2. *How can we scale the proposed approach to real-world programs, properties, and users?*

I addressed these challenges by proposing a unified constraint-based framework for user-centric program analysis. For separation of concerns, the framework comprises a front-end, Petablox, and a back-end, Nichrome, which address the above two challenges respectively.

Petablox[1] addresses the first challenge for arbitrary program analyses specified in Datalog, a declarative logic programming language. Instead of using a fixed abstraction, Petablox synthesizes a family of abstractions that differ in accuracy, scalability, or sometimes soundness, and dynamically selects an abstraction that is most suitable for the current usage scenario. I refer to this problem of adapting the analysis to a given usage scenario as the *user-centric analysis problem.* Petablox formulates this problem as a system of mixed hard and soft constraints. While the hard constraints encode the family of viable abstractions, the soft constraints encode the objective of finding the optimum abstraction under the given usage scenario by balancing various tradeoffs.

Compared to the conventional analysis problem, which is a satisfiability problem, the user-centric analysis problem is even more challenging to solve as it is an optimization problem. Ideally, a solver for such problems should be *sound* (i.e., it does not violate any hard constraint), *optimal*, (i.e., it maximizes the objective), and *scalable* (i.e., it can solve constraints generated from real-world programs, properties, and users). All existing solvers such as Alchemy, Tuffy, RockIt, CPI, and Z3 sacrifice one or more of the three properties. To address this challenge, Nichrome[2] solves a system of mixed hard and soft constraints by reducing it into a (weighted partial) *maximum satisfiability* (MaxSAT) problem. To enable sound, optimal, and scalable solving, I have proposed several novel techniques in both problem reduction and MaxSAT solving. While Nichrome successfully solves the user-centric analysis problem, it is not limited to program analysis and can be applied to problems in other domains like information retrieval, machine learning, and mathematical optimization.

Next, I elaborate upon Petablox and Nichrome.

Petablox: A Unified Framework for Adaptive Program Analysis

Petablox accepts analyses specified in Datalog which is widely adopted by the program analysis community for specifying analyses. Since it is declarative, Datalog allows the designer to focus on high-level analysis specifications,

[1] http://petablox.org

[2] http://nichrome.org

rather than low-level implementation details, which are handled by powerful off-the-shelf Datalog solvers. More importantly, by analyzing the Datalog constraints, we obtain a general and uniform mechanism to make any analysis specified in Datalog user-centric.

To make an analysis user-centric, I identified three concrete targets in a usage scenario to adapt the abstraction to: feedback from analysis users, assertions of interest, and characteristics of the program at hand. By adapting to these targets, Petablox synthesizes an abstraction that is optimal in terms of soundness, accuracy, and scalability for the given usage scenario. Petablox formulates the problem of adapting an analysis to a given target as a system of mixed hard and soft constraints: while the hard constraints for all three targets encode the soundness conditions and thereby define the space of viable abstractions, the soft constraints balance different tradeoffs, and thereby define the objective of selecting the most suitable abstraction for the given target. We next describe how Petablox adapts the analysis to each target in detail.

Adapting to User Feedback [TR'17a, FSE'15]. In practice, program analysis tools often produce many false alarms, which imposes a steep burden on the analysis user. The key issue is that when the designer chooses the abstraction, their notion about usefulness of the analysis results fails to match the user's notion. By adapting the abstraction to user feedback, Petablox shifts this decision about usefulness of analysis results from the designer to the user, and thereby reduces the false alarm rate which in turn improves the accuracy of the analysis.

Following the above idea, we recently proposed an iterative and interactive approach to resolve analysis alarms [TR'17a]. In each iteration, our approach identifies a set of potential common root causes of false alarms by applying unsound but precise heuristics that are extracted from practical analysis tools. Instead of directly suppressing these root causes, which may unsoundly eliminate alarms, it poses questions to the user about their truthhood. If the user confirms them, only then are they suppressed, thereby eliminating all false alarms resulting from them, with the user's knowledge. There are two features that makes our approach effective: 1) it is able to resolve a large number of false alarms by asking a few questions due to the observation that most alarms are often symptoms of a relatively small set of common root causes; 2) by posing questions iteratively, it is able to prioritize questions with high payoff in early iterations. To realize these two features, the approach iteratively solves a user-centric analysis problem that maximizes the benefit-cost ratio in terms of the number of false alarms expected to be eliminated and the number of questions posed. As a result, the soft constraints in the generated problem balance the tradeoff between the benefits and cost of each posed question. We empirically showed that this approach is able to eliminate 68% of the false alarms with an average payoff of 10× per question. Moreover, it prioritizes user effort effectively by posing questions that yield high payoff earlier.

Our earlier work [FSE'15], which won an **ACM SIGSOFT Distinguished Paper Award**, incorporates user feedback on end reports as compared to feedback on intermediate results. By considering both positive and negative feedback from the user, it suppresses alarms similar to the ones that the user dislikes while keeping alarms similar to the ones they like. Our approach achieves this effect by making the analysis probabilistic. It attaches a weight to each analysis rule, representing confidence, such that the rules known to often cause false alarms get lower weights. These weights can be chosen by the designer manually or learnt automatically using labeled data. By attaching weights, we transform analysis rules from hard rules into soft rules, which can be violated based on user feedback to yield different abstractions. As a result, the probabilistic analysis defines a set of analysis outputs rather than a single output, and the corresponding user-centric analysis problem aims to find the output that most likely matches the user's preferences. Compared to the previous approach, this approach does not require the user to inspect intermediate analysis results, but it can introduce false negatives and thereby render the result unsound. The soft constraints in the corresponding user-centric analysis problem balance the tradeoff between soundness and completeness. We empirically showed that, on average our approach can eliminate 70% of the false alarms by labeling 5% of the alarms at the cost of introducing 2% false negatives.

Adapting to Assertions of Interest [PLDI'13, PLDI'14a]. A central challenge in designing a program analysis is to balance the tradeoff between accuracy and scalability. While a fine-grained abstraction can resolve more assertions but cannot scale to large programs, a coarse-grained abstraction can scale to larger programs but resolve fewer assertions. An observation commonly exploited by query-driven or demand-driven program analysis is that, to resolve a given assertion, we often only need to analyze a small relevant part of the whole program. Thus, by tailoring the abstraction to a given assertion of interest, we can potentially generate an abstraction that is both accurate and efficient. Unfortunately, existing approaches only target specific analyses and none of them guarantee the optimality of the generated abstraction in terms of efficiency.

To address this challenge, we proposed a unified approach to infer an *optimum* abstraction to resolve a given assertion in a program for any analysis specified in Datalog. This approach is described in two consecutive works [PLDI'13, PLDI'14a], one of which won an **ACM SIGPLAN Distinguished Paper Award** [PLDI'14a]. The input to our approach is a program, an assertion of interest, and a Datalog analysis parametrized by a family of abstractions which are partially ordered by their precision and cost. The approach either finds a cheapest abstraction that can prove the assertion or concludes that there is no viable abstraction in the family. In the corresponding user-centric analysis problem, while the hard constraints encode the space of viable abstractions that can resolve the given assertion, the soft constraints encode the objective to minimize the analysis cost. We empirically showed that our approach successfully resolves all assertions on large Java programs of the order of hundreds of thousands of lines of code, while the baseline approach which applies a uniform expensive abstraction only resolves upto 50% of the assertions and consumes significantly more time.

Adapting to Program Characteristics [PLDI'14b, OOPSLA'16]. By exploiting the characteristics of the program at hand, we can specialize the analysis abstraction to the program, therefore improving the effectiveness. One important program characteristic to adapt to, is the pattern of procedure reuses as the accuracy and scalability of an analysis is greatly impacted by how it handles procedure reuses. In Petablox, I have adapted the analysis abstraction to procedure reuses both within a program and across programs that share the same library code.

Context-sensitivity, the ability to distinguish analysis results in each procedure under different calling contexts, is crucial to an analysis's accuracy. To achieve context-sensitivity in a scalable manner, interprocedural analyses typically compute summaries that capture the results for each procedure under different calling contexts and instantiate them when the calling context matches. A key challenge in constructing a summary is to balance the tradeoff between specialization and generalization. Current summary-based interprocedural analyses can be broadly classified into top-down analyses and bottom-up analyses. Top-down analyses compute summaries that are highly specific to a given calling context, making it efficient to compute and instantiate but less reusable. As a result, it can hinder the scalability by computing an excessive amount of summaries. In contrary, bottom-up analyses compute summaries that are applicable to all calling contexts, making them highly reusable but potentially expensive to compute and apply, therefore also hindering the scalability. In [PLDI'14b], we proposed a hybrid approach that combines the benefit of these two approaches without their drawbacks. By identifying the most common calling contexts of a given procedure, it computes a summary that synergistically combines top-down and bottom-up summaries, which balances the tradeoff between specialization and generalization. Our empirical evaluation shows that our approach achieves speedups upto 59× over the top-down approach and 118× over the bottom-up approach.

Our recent work targets reusing results across programs rather than within a program. It is motivated by the observation that practical programs share large modules of code. Although the implementations of these modules are the same across programs, their usages differ. To address this challenge, similar to the previous approach, we generate procedure summaries for library APIs by solving user-centric analysis problems that balance the tradeoff between specialization and generalization. On one hand, a highly specialized summary can lead to significant efficiency benefit by aggressively pruning away intermediate results, but is less reusable. On the other hand, a very general summary is highly reusable but may only lead to modest cost reduction. By solving the corresponding user-centric analysis problem, the summaries generated by our approach strike a sweet spot between these two extremes. We showed empirically that our approach achieves average speedups of 2.6× and 5.2× for two foundational analyses of large Java programs.

Nichrome: A Scalable Solver for Weighted Constraints

Nichrome solves different user-centric analysis problems generated by Petablox in a unified manner. Its input is a system of weighted constraints, where the hard constraints are conventional Datalog rules, and the soft constraints are Datalog rules augmented with weights. It outputs a solution that satisfies all hard rule instances while maximizing the sum of weights of all satisfied soft rule instances. It finds such a solution by employing a two-phase approach: it first reduces the input problem into a MaxSAT problem and then solves it by leveraging an off-the-shelf MaxSAT solver. To solve the problem of weighted constraints in a sound, precise, and scalable manner, I proposed novel techniques in both phases, which I elaborate below.

Lazy-Eager Grounding [SAT'15, AAAI'16]. The grounding phase reduces weighted constraints into a MaxSAT formula. A naive approach which replaces the quantified variables with possible valuations can result in an exponential blowup in the size of the generated formula. For certain user-centric analysis problems, such formulae can contain up to 10^{30} clauses, which are intractable for any existing MaxSAT solver. In [SAT'15, AAAI'16], we proposed a lazy-eager iterative grounding technique. It is lazy in the sense that, instead of fully grounding all clauses using the naive approach, it only poses a subset of the clauses to the MaxSAT solver in each iteration. Our approach terminates when it deems that the solution returned by the solver is indeed a solution to the full problem; otherwise, it refines the set of considered clauses by adding more clauses to it. It is eager in the sense that we compute the initial set of ground clauses by leveraging the least fixpoint semantics of the original Datalog program, therefore reducing the number of iterations consumed by our approach. Our approach successfully reduces the number of clauses in the largest MaxSAT formula from 10^{30} to 10^7. Though the sizes of the generated formulae have been significantly reduced, the larger instances are still beyond the scope of state-of-the-art solvers, which motivates our innovations in MaxSAT solving.

Query-Guided and Incremental MaxSAT Solving [POPL'16, CP'16]. To scale MaxSAT solvers to the above large instances, we proposed two novel solving techniques which can be also applied to general MaxSAT instances.

In [POPL'16] we defined a new optimization problem *query-guided maximum satisfiability*, or Q-MaxSAT. The key insight behind Q-MaxSAT is that, for many MaxSAT instances generated from practical applications, one is interested in small set of *queries* that constitute a very small fraction of the entire MaxSAT solution. For instance, in program analysis, a query could be analysis information for a particular variable in the program—intuitively, one would expect the computational cost for answering a small set of queries to be much smaller than the cost of computing analysis information for all program variables. Following this insight, we developed a novel iterative algorithm to solve the Q-MaxSAT problem. We evaluated this approach on 19 Q-MaxSAT instances generated from program analysis and information retrieval problems comprising upto 2×10^7 clauses each. Our approach solved all instances with modest time and memory consumption while conventional MaxSAT solvers failed to terminate on 8 of them.

In [CP'16], we proposed *incremental MaxSAT solving.* Many emerging applications including our aforementioned grounding algorithm involve solving a sequence of similar MaxSAT instances. To improve the overall efficiency, it is desirable to share the results across instances in the same sequence rather than solving each from scratch. Following this idea, we developed an incremental algorithm based on the popular UNSAT-core-guided MaxSAT algorithm. We evaluated our approach on 74 sequences of large MaxSAT instances generated from diverse applications in program analysis and information retrieval. We showed that our approach yields an average speedup of 1.8× per sequence over a state-of-the-art non-incremental solver, and solves 19 more sequences.

Future Directions

I plan to extend Petablox and Nichrome in both applications and algorithms along the following directions:

Effective Detection of Security Vulnerabilities. I intend to apply Petablox to help detect security vulnerabilities in real-world programs. Program analysis tools have made remarkable strides in finding bugs related to security vulnerabilities. However, they often fail to realize their full potential due to high false positive rates. For instance, APISan, a tool recently proposed for detecting API usage errors, finds 76 previously unknown bugs in widely-used software including Linux kernel, OpenSSL, PHP, and Python interpreter. Such bugs include ones that can tamper memory safety and in turn introduce security vulnerabilities. However, the authors only managed to inspect a small fraction of the total 40,000 reports due to the high false positive rate (> 80%), leaving more bugs lurking in the remaining reports. Starting with APISan whose authors are also from Georgia Tech, I plan to apply Petablox to help sift true reports from false alarms produced by these tools in an effective and effortless manner, therefore uncovering more security vulnerabilities.

Learning Program Analyses from Big Code. I expect Petablox to adapt a program analysis to its usage scenarios even more effectively by leveraging Big Code. Inspired by the success of Big Data analytics, Big Code aims at improving program reasoning by identifying and understanding commonalities among the large body of open source software available today. Big Code can help improve the effectiveness of Petablox in two ways. First, instead of relying on expert designers to define the analysis specifications, we can automatically learn them from existing codebases and bug reports. One concrete direction is to synthesize Datalog rules using existing programs and their known bug reports as the input and output. Second, by learning from past Petablox runs within the same program and across multiple programs, we can speedup the process of solving the adaptivity problems. For instance, when adapting to user feedback, we can predict potential root causes of false alarms more effectively by learning from past user feedback. This in turn can reduce the iterations required to achieve similar false alarm reduction by avoiding asking unnecessary questions.

Program Partitioning on Heterogeneous Platforms. Besides adapting program analyses, Nichrome can also be applied to solve constraint problems that adapt programs themselves. One example application is mobile-cloud computing, which has emerged as a new paradigm to augment a resource-constrained device by offloading its computation to a more powerful device. Such a device can be a mobile phone, a smart watch, or an embedded device in the "Internet of Things", while the more powerful device can be a remote server, home desktop, personal laptop, or a tablet. A key challenge in partitioning a mobile-cloud program between two ends is that the communication cost in time and energy should not offset the computation gain. Though the application seems irrelevant to user-centric program analysis, interestingly, the underlying problem formulation is similar, which can also be cast as a system of weighted constraints. While the hard constraints enforce the semantics of the unoffloaded execution and address security and privacy concerns, the soft constraints balance the tradeoff between computation and communication. Following this insight, I recently submitted a paper for partitioning mobile-cloud programs using Nichrome [TR'17b].

A Scalable Solver for Problems Beyond NP. I plan to extend Nichrome to richer problem formulations and investigate new solving techniques for these problems. Constraint-based approaches have emerged as a new computational paradigm not only in program analysis but all areas of computer science over the past few decades, which is exemplified by the use of SAT solvers to tackle problems in the complexity class NP. However, with the presence of even more demanding applications like probabilistic reasoning in machine learning, programming languages, and databases, there is a demand for solvers that reach beyond NP [3]. The constraint problems targeted by such solvers include maximum satisfiability (MaxSAT), maximum satisfiability modulo theories (MaxSMT), integer linear programming (ILP), and quantified boolean formula (QBF). State-of-the-art solvers for these problems are much less scalable compared to SAT solvers. I plan to address this challenge by extending Nichrome to these problems. In addition to applying *query-guided* solving techniques and *incremental* solving techniques, motivated by the success of modular program analysis, I will investigate *compositional* solving techniques, which further improve the solver performance by breaking the constraint problem into more tractable subproblems.

References

[TR'17a] X. Zhang, R. Grigore, X. Si, and M. Naik. Effective interactive resolution of static analysis alarms. 2016.

[TR'17b] X. Si, X. Zhang, H. Esmaeilzadeh, and M. Naik. Exploring computation-communication tradeoffs in mobile-cloud computing via graph partitioning. 2016.

[POPL'16] X. Zhang, R. Mangal, A. V. Nori, and M. Naik. Query-guided maximum satisfiability. In *POPL*, 2016.

[OOPSLA'16] S. Kulkarni, R. Mangal, X. Zhang, and M. Naik. Accelerating program analyses by cross-program training. In *OOPSLA*, 2016.

[AAAI'16] R. Mangal, X. Zhang, A. Kamath, A. V. Nori, and M. Naik. Scaling relational inference using proofs and refutations. In *AAAI*, 2016.

[CP'16] X. Si, X. Zhang, V. M. Manquinho, M. Janota, A. Ignatiev, and M. Naik. On incremental core-guided maxsat solving. In *CP*, 2016.

[FSE'15] R. Mangal, X. Zhang, A. V. Nori, and M. Naik. A user-guided approach to program analysis. In *FSE*, 2015.

[SAT'15] R. Mangal, X. Zhang, A. V. Nori, and M. Naik. Volt: A lazy grounding framework for solving very large maxsat instances. In *SAT*, 2015.

[PLDI'14a] X. Zhang, R. Mangal, R. Grigore, M. Naik, and H. Yang. On abstraction refinement for program analyses in datalog. In *PLDI*, 2014.

[PLDI'14b] X. Zhang, R. Mangal, M. Naik, and H. Yang. Hybrid top-down and bottom-up interprocedural analysis. In *PLDI*, 2014.

[PLDI'13] X. Zhang, M. Naik, and H. Yang. Finding optimum abstractions in parametric dataflow analysis. In *PLDI*, 2013.

[3] `http://beyondnp.org`