Viktor Kuncak: Research Statement

The goal of my research is to develop software analysis techniques that improve software reliability. The success of specialized program analyses, finite state verification, and type systems demonstrates that, while the general program analysis problem may be very challenging, the use of specific, targeted solutions can be extremely effective. As memory-safe programming languages eliminate the sources of low-level errors, the focus is shifting to deeper semantic properties.

Building on these observations, my research approach uses multiple specialized techniques within a unified analysis framework, with each technique solving an appropriate subset of the overall analysis problem. This approach involves identifying relevant deep properties that are beyond the reach of current analyses, designing and implementing new analysis algorithms for these properties, and deploying these algorithms in the context of a unified analysis framework. Deep properties require precise and therefore unscalable analyses. The framework enables the targeted application of such analyses to small regions of code; it then combines the results of the individual analyses to prove properties of the entire program. A unified framework also solves a number of issues in applying specialized analyses to verify deep semantic properties: scalability through data abstraction, communicating information between analyses, establishing analysis soundness, and the ability for the developer to direct the analysis. Compared to any individual analysis or any monolithic reasoning approach, the synergy of specialized analyses enables more automated verification of more sophisticated properties of interest. This approach blurs the boundaries between **programming languages** (program analysis and language design), **software engineering** (design by contract and separation of concerns), and **formal methods** (software model checking, theorem proving, and decision procedures). It provides a road map to building systems that have increasingly sophisticated internals and can verify increasingly expressive classes of properties, but retain a clean, stable interface to the software developer.

Thesis research. In my thesis research I apply this approach to reason about linked mutable data structures, analyzing their implementation, their usage protocols, and the relationships between different data structures within a program. My work develops new analysis techniques that can analyze new classes of program properties, shows how to combine these techniques into a unified framework, and develops systems based on these results. My motivation for this research is that software with linked data structures is both important and difficult to reason about. It is important because linked data structures give the flexibility needed for many efficient algorithms, and are common in many Java and C programs. Unfortunately, this same flexibility makes it easy for programmers to introduce errors, ultimately causing software to crash or produce unexpected results. Automated reasoning about such software is therefore desirable, but the flexibility of linked data structures implies that general purpose techniques will always remain either imprecise or unscalable.

To address this problem I have developed the Hob and Jahob systems for verifying data structures. Hob [5] allows developers to specify data structure interfaces using abstract sets of objects. Hob uses a new shape analysis [16] to verify, with new levels of predictability and automation, that implementations of linked data structures such as binary trees conform to desirable set interfaces. The theorem proving plugin of Hob [18] enables the verification of arbitrarily complex data structures using interactive theorem proving. A new generalized typestate analysis [14] enables Hob to analyze usage protocols for data structures whose implementations are verified using shape analysis and theorem proving, and to analyze relationships between data structures, such as disjointness, inclusion, and equality. By combining different analyses, Hob enables precise analyses to produce results meaningful in the context of a larger program without applying these expensive techniques to the entire program, and enables generalizations of typestate analyses to correlate complex properties that could not be analyzed by typestate analyses alone. I am currently developing a new system, Jahob [7, 12], which generalizes the ideas of expressive module interfaces to a richer specification language based on relations, and whose architecture and reasoning techniques enable it to analyze a significantly broader range of data structures and usage patterns than is possible in Hob.

Future directions. We live in exciting times for software analysis, with advances in analysis algorithms enabling the verification of new program properties, with modular verification techniques enabling the application of these algorithms to relevant programs, and with industry starting to recognize the importance of program specifications that support such modular verification. This situation makes me believe that our modular analysis techniques have the potential for a significant impact on the way software is developed. To realize this potential, I expect to increase the usability of static analysis systems by integrating them with techniques for finding bugs in specifications and code based on counterexample generation [3]. I will also explore a number of techniques for making code annotation easier. I plan to verify data structures consistency properties in operating systems [1, 17] and computer games [14].

In the future I plan to extend the analysis system with techniques for reasoning about software designs and software architectures. My experience in modelling air-traffic control software [21] and cryptographic protocols [23] suggests that automated reasoning about software models can fundamentally improve the architecture of software systems. I intend to build systems that support the creation and exploration of software models as well as the transformation of models into working software. I expect this research to result in techniques that will enable us to build new generations of software systems that are more reliable, secure, and sophisticated than any system we can hope to construct today.

Verifying Data Structure Consistency: Recent and Ongoing Work

My thesis research develops algorithms for reasoning about data structures that programs manipulate, and evaluates these algorithms in the context of the Hob and Jahob analysis systems that I am developing. In Hob and Jahob, the developer specifies data structure interfaces using sets and relations. The system then uses an appropriate analysis technique to verify each data structure implementation. Finally, the system uses a potentially different analysis technique to verify data structure usage as well as high-level relationships between data structures. The system uses data abstraction to separate the analysis of data structure implementation from the analysis of data structure clients: data structure interfaces are written in terms of abstract state variables that do not reveal implementation details, and are connected to concrete state using private abstraction functions.

The **Hob system** [5, 14, 18] contains an analyzer for a memory-safe imperative language with dynamic memory allocation and direct support for data abstraction. When writing programs in Hob, developers specify procedure interfaces using abstract sets of objects. Our experience suggests that this set abstraction works well as a technique for decomposing the analysis tasks: we were successful in analyzing data structure implementations using symbolic shape analysis [16] and analyzing relationships between data structures using an analysis that manipulates formulas with set variables [14]. We verified applications such as a minesweeper game implementation, a water particle simulation, and a web server [13].

I am currently developing the **Jahob system** for verifying data structures in programs in a subset of Java. As a specification language for data structure interfaces Jahob uses a subset of the formulas of the Isabelle interactive theorem prover. By choosing popular and general-purpose implementation and specification languages, my goal is to increase the future adoption and impact of the system. These design decisions will also make it easier to extend the system to verify additional properties, using the experience I gained at Microsoft Research [6] on ESC/Java-like systems [19]. I am currently focusing on verifying data structure specifications that generalize Hob's set interfaces and the underlying algorithms with 1) size constraints on data structures and 2) relations that describe maps or an unbounded number of data structures.

My results on algorithms for solving size constraints on sets are a natural outgrowth of studying the language of sets that describe global data structures. Size constraints on such sets allow the system to verify, for example, that a size field of a data structure is consistent with the actual number of elements stored in the data structure; neither Hob nor any other system we know of can analyze such constraints in an automated way. Using my previous experience in solving an open problem about structural subtyping [9, 10], I have formalized an algorithm for deciding quantified formulas of sets with size constraints. I gave the first complexity analysis and the first implementation of the algorithm [7], settling some recently posed open problems [24, 22]. Using the Jahob system we have demonstrated that a combination of our algorithm with the set abstraction enables the verification of size fields of data structures. Driven by the experience from our implementation, I have recently examined the quantifier-free fragments of these constraints and found a polynomial space algorithm [15] (previous algorithms had exponential time and space complexity). As part of the subsequent ongoing work, we have characterized several NP-complete fragments and identified a polynomial-time fragment [15], establishing a trade-off between the expressive power and the complexity of reasoning; we plan to use these results to construct efficient analyses for checking generalized typestate properties as well as relationships between data structures.

Relational data structure interfaces in Jahob provide two key generalizations over the interfaces in Hob: 1) they can describe the high-level functionality of association lists, hash tables, and sorted trees that cannot be described as sets of objects, but can be described as relations between keys and values, and 2) they can specify not only a statically bounded number of global data structures as in Hob, but also data structures that can be dynamically instantiated [12] which is the common way of implementing data structures in Java. These generalizations lead to several challenges that I am addressing in Jahob. Verifying that a data structure conforms to a map specification produces more complex proof obligations; I am using fine-grained data abstraction to discharge such proof obligations. Relational interfaces also require a sophisticated analysis of relationships between data structures; I am developing such analysis using ideas from role analysis [4, 8, 11] and an approach to dynamically changing ownership between objects. In the future I would also like to support data structure interfaces that use algebraic data types, building on the positive experience in proving properties of purely functional data structures in Isabelle [2], and the integration of algebraic data types into the framework of relational reasoning [3].

Future Work: Usability and Impact of Analysis Systems

I believe that the approach behind the Hob and Jahob systems has the potential to substantially improve software reliability. To investigate this claim, I plan to apply such analysis systems to software used on a daily basis, such as operating systems and computer games. In the context of operating systems, I am particularly interested in file systems; I have previously explored the use of interactive theorem provers for proving properties of file system models [1, 17] and obtained promising initial results on verifying file system components using Jahob. I am also interested in enforcing security properties of code by both verifying the protocols themselves [23] and verifying that the code implements the protocols correctly [20].

To make it easier to apply the analysis system to larger pieces of software, I would like to improve the system's ability to quickly identify errors and present concrete feedback in the form of counterexamples. For this purpose I plan to use counterexample generation algorithms and finite model finding techniques extended with the ability to handle concrete domains such as algebraic data types [3]. I will consider new techniques for underapproximating code that improve the state space coverage as compared to loop unrolling approaches.

My experience in developing verified programs in Hob and Jahob suggests that a useful system should provide both 1) the possibility to infer program annotations automatically, and 2) the possibility for the developer to insert such annotations manually, in order to debug the verification process, overcome any limitations of the analysis, and ultimately specify the desired correctness properties. I am particularly interested in new types of program annotations that provide partial information to the analysis, capturing subtle aspects of the implementation while allowing the analysis to infer more straightforward details. I am also interested in reducing the size of annotations through language constructs that factor out the common fragments of annotations in a program [14]. In addition to using specifications expressed as formulas, I will investigate the use of test suites as partial specifications. Finally, I believe that there is still a large space of unexplored techniques for synthesizing loop invariants and procedure contracts for data structure properties; I plan to explore combinations of semantic approaches such as predicate abstraction and symbolic shape analysis [16] with domain-specific and syntactic approaches such as generalized typestate analysis [14]. I am also interested in exploring the impact of dynamically collected information on static inference of annotations. I expect that the problem of combining these techniques will present new research challenges, whose solution will result in systems that are cost-effective and appealing to software developers.

Future Work: Design and Implementation of Dependable Systems

In addition to building program analysis systems, I am interested in using automated reasoning for software design and modelling. I would like to build systems that help developers in writing software models, rapidly exploring software architecture alternatives, and transforming models into implementations. My experience in modelling air-traffic control applications [21] and cryptographic protocols [23] has convinced me that software models are very valuable in the context of software development. I hope to make models even more valuable by connecting them to analysis systems with data abstraction capabilities, which will help developers establish the conformance of their code to the model, and subsequently maintain this conformance as the software evolves. In addition, I would like to develop techniques for efficiently maintaining models during program execution and exposing these models to the outside world. I expect that giving developers and users access to such information will significantly improve the interfaces of software components and foster the construction of more sophisticated software systems.

I believe that the long-term progress of software development systems will depend on their ability to incorporate high-level knowledge that human developers use today when creating software. This knowledge includes not only the knowledge of programming languages, algorithms, and data structures, but also the knowledge of a particular domain. In the future I plan to build software development environments that use such domain-specific knowledge to make informed decisions about the analysis and transformation of software models and implementations. One area where this approach can have a large impact is the domain of embedded and safety-critical systems with real-time constraints. I will introduce language constructs that make it easier to express and analyze such systems. Building on my work on shape analysis [16] and data structure size analysis [7], I will develop flexible memory management solutions with statically computable timing behavior and incorporate techniques for automated reasoning about timing constraints. To enable modelling uncertainty in the environment and provide a larger space for abstracting system's details, I intend to use probabilistic extensions of logics with sets and relations, and explore the corresponding reasoning techniques. I expect such domain-specific systems to dramatically reduce the resources needed to develop reliable and sophisticated software for these challenging domains.

Referenced publications I have coauthored (see CV for full list)

[1] Konstantine Arkoudas, Karen Zee, Viktor Kuncak, and Martin Rinard. Verifying a file system implementation. In *Sixth International Conference on Formal Engineering Methods (ICFEM'04)*, volume 3308 of *LNCS*, Seattle, Nov 8-12, 2004 2004.

[2] Viktor Kuncak. Binary search trees. The Archive of Formal Proofs, `http://afp.sourceforge.net/`, April 2004.

[3] Viktor Kuncak and Daniel Jackson. Relational analysis of algebraic datatypes. In *Joint 10th European Software Engineering Conference (ESEC) and 13th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, 2005.

[4] Viktor Kuncak, Patrick Lam, and Martin Rinard. Role analysis. In *Annual ACM Symp. on Principles of Programming Languages (POPL)*, 2002.

[5] Viktor Kuncak, Patrick Lam, Karen Zee, and Martin Rinard. Implications of a data structure consistency checking system. In *Int. conf. on Verified Software: Theories, Tools, Experiments (VSTTE, IFIP Working Group 2.3 Conference)*, Zürich, October 2005.

[6] Viktor Kuncak and K. Rustan M. Leino. In-place refinement for effect checking. In *Second International Workshop on Automated Verification of Infinite-State Systems (AVIS'03), Warsaw, Poland*, April 2003.

[7] Viktor Kuncak, Hai Huu Nguyen, and Martin Rinard. An algorithm for deciding BAPA: Boolean Algebra with Presburger Arithmetic. In *20th International Conference on Automated Deduction, CADE-20*, Tallinn, Estonia, July 2005.

[8] Viktor Kuncak and Martin Rinard. Existential heap abstraction entailment is undecidable. In *10th Annual International Static Analysis Symposium (SAS 2003)*, San Diego, California, June 11-13 2003.

[9] Viktor Kuncak and Martin Rinard. On the theory of structural subtyping. Technical Report 879, Laboratory for Computer Science, Massachusetts Institute of Technology, 2003.

[10] Viktor Kuncak and Martin Rinard. Structural subtyping of non-recursive types is decidable. In *Eighteenth Annual IEEE Symposium on Logic in Computer Science*, 2003.

[11] Viktor Kuncak and Martin Rinard. Generalized records and spatial conjunction in role logic. In *11th Annual International Static Analysis Symposium (SAS'04)*, Verona, Italy, August 26–28 2004.

[12] Viktor Kuncak and Martin Rinard. Decision procedures for set-valued fields. In *1st International Workshop on Abstract Interpretation of Object-Oriented Languages (AIOOL 2005)*, 2005.

[13] Patrick Lam, Viktor Kuncak, and Martin Rinard. On our experience with modular pluggable analyses. Technical Report 965, MIT CSAIL, September 2004.

[14] Patrick Lam, Viktor Kuncak, and Martin Rinard. Generalized typestate checking for data structure consistency. In *6th International Conference on Verification, Model Checking and Abstract Interpretation*, 2005.

[15] Bruno Marnette, Viktor Kuncak, and Martin Rinard. On algorithms and complexity for sets with cardinality constraints. Technical report, MIT CSAIL, August 2005.

[16] Thomas Wies, Viktor Kuncak, Patrick Lam, Andreas Podelski, and Martin Rinard. Field constraint analysis. In *Proc. Int. Conf. Verification, Model Checking, and Abstract Interpratation*, 2006.

[17] Karen Zee and Viktor Kuncak. File refinement. The Archive of Formal Proofs, `http://afp.sourceforge.net/`, December 2004.

[18] Karen Zee, Patrick Lam, Viktor Kuncak, and Martin Rinard. Combining theorem proving with static analysis for data structure consistency. In *International Workshop on Software Verification and Validation (SVV 2004)*, Seattle, November 2004.

Further references

[19] Cormac Flanagan, K. Rustan M. Leino, Mark Lilibridge, Greg Nelson, James B. Saxe, and Raymie Stata. Extended Static Checking for Java. In *ACM Conf. Programming Language Design and Implementation (PLDI)*, 2002.

[20] Jean Goubault-Larrecq and Fabrice Parrennes. Cryptographic protocol analysis on real C code. In Radhia Cousot, editor, *Proc. Int. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, 2005.

[21] Daniel Jackson, Martin Rinard, John Hansman, and David Schmidt. ITR: Design conformant software (0086154), Annual project report, July 2003.

[22] Peter Revesz. Quantifier-elimination for the first-order theory of boolean algebras with linear cardinality constraints. In *Proc. Advances in Databases and Information Systems (ADBIS'04)*, volume 3255 of *LNCS*, September 2004.

[23] Emina Torlak, Marten van Dijk, Blaise Gassend, Daniel Jackson, and Srinivas Devadas. Knowledge flow analysis for security protocols. Technical Report MIT-CSAIL-TR-2005-066, MIT-LCS-TR-1007, Computer Science and AI Lab, 2005.

[24] Calogero G. Zarba. A quantifier elimination algorithm for a fragment of set theory involving the cardinality operator. In *18th International Workshop on Unification*, 2004.