Seminar: Aktuelle Themen aus der Theorie der Programmierung

Numerical Domains

A Practical Construction for Decomposing Numerical Abstract Domains

Gagandeep Singh, Markus Püschel, Martin Vechev

Numerical abstract domains such as Polyhedra, Octahedron, Octagon, Interval, and others are an essential component of static program analysis. The choice of domain offers a performance/precision tradeoff ranging from cheap and imprecise (Interval) to expensive and precise (Polyhedra). Recently, significant speedups were achieved for Octagon and Polyhedra by manually decomposing their transformers to work with the Cartesian product of projections associated with partitions of the variable set. While practically useful, this manual process is extremely time consuming, error-prone and has to be applied from scratch for every domain.

In this paper, we present a novel approach that can soundly decompose any sub-polyhedra domain. Unlike prior work, the method is generic in nature and does not require changes to the original abstract transformers or additional manual effort per domain. Further, it presents guarantees on the partitions achievable by each decomposed transformer. In general, our method achieves finer partitions than prior work.

We implemented our approach and applied it to the domains of Zones, Octagon, and Polyhedra. We then compared the performance of the decomposed transformers obtained with our generic method versus state-of-the art PPL and the faster ELINA (which uses manual decomposition). Against the latter we demonstrate finer partitions and an associated speedup of about 2x on average. Our results indicate that the construction presented in this work is a viable method for improving the performance of numerical domains. It enables designers of abstract domains to benefit from decomposition without re-writing all of their transformers from scratch (as required by prior methods). https://doi.org/10.1145/3158143

Fast Polyhedra Abstract Domain

Gagandeep Singh, Markus Puschel, Martin Vechev

Numerical abstract domains are an important ingredient of modern static analyzers used for verifying critical program properties (e.g., absence of buffer overflow or memory safety). Among the many numerical domains introduced over the years, Polyhedra is the most expressive one, but also the most expensive: it has worst-case exponential space and time complexity. As a consequence, static analysis with the Polyhedra domain is thought to be impractical when applied to large scale, real world programs. In this paper, we present a new approach and a complete implementation for speeding up Polyhedra domain analysis. Our approach does not lose precision, and for many practical cases, is orders of magnitude faster than state-of-the-art solutions. The key insight underlying our work is that polyhedra arising during analysis can usually be kept decomposed, thus considerably reducing the overall complexity. We first present the theory underlying our approach, which identifies the interaction between partitions of variables and domain operators. Based on the theory we develop new algorithms for these operators that work with decomposed polyhedra. We implemented these algorithms using the same interface as existing libraries, thus enabling static analyzers to use our implementation with little effort. In our evaluation, we analyze large benchmarks from the popular software verification competition, including Linux device drivers with over 50K lines of code. Our experimental results demonstrate massive gains in both space and time: we show endto-end speedups of two to five orders of magnitude compared to state-of-the-art Polyhedra implementations as well as significant memory gains, on all larger benchmarks. In fact, in many cases our analysis terminates in seconds where prior code runs out of memory or times out after 4 hours. We believe this work is an important step in making the Polyhedra abstract domain both feasible and practically usable for handling large, real-world programs. http://dl.acm.org/authorize?N29913

Sound Bit-Precise Numerical Domains

Tushar Sharma, Thomas Reps

This paper tackles the challenge of creating a numerical abstract domain that can identify affine-inequality invariants while handling overflow in arithmetic operations over bit-vector data-types. The paper describes the design and implementation of a class of new abstract domains, called the Bit-Vector-Sound, Finite-Disjunctive (BVSFD) domains. We introduce a framework that takes an abstract domain A that is sound with respect to mathematical integers and creates an abstract domain BVS(A) whose operations and abstract transformers are sound with respect to machine integers. We also describe how to create abstract transformers for BVS(A) that are sound with respect to machine arithmetic. The abstract transformers make use of an operation WRAP(av,v) −where av∈A and v is a set of program variables−which performs wraparound in av for the variables in v.

To reduce the loss of precision from WRAP , we use finite disjunctions of BVS(A) values. The constructor of finite-disjunctive domains, FDk(·) , is parameterized by k, the maximum number of disjunctions allowed.

We instantiate the BVS(FDk) framework using the abstract domain of polyhedra and octagons. Our experiments show that the analysis can prove 25% of the assertions in the SVCOMP loop benchmarks with k=6, and 88% of the array-bounds checks in the SVCOMP array benchmarks with k=4 .

http://link.springer.com/chapter/10.1007/978-3-319-52234-0_27

Concurrency (Parallelität)

Thread modularity at many levels: a pearl in compositional verification

Jochen Hoenicke, Rupak Majumdar, Andreas Podelski

A thread-modular proof for the correctness of a concurrent program is based on an inductive and interference-free annotation of each thread. It is well-known that the corresponding proof system is not complete (unless one adds auxiliary variables). We describe a hierarchy of proof systems where each level k corresponds to a generalized notion of thread modularity (level 1 corresponds to the original notion). Each level is strictly more expressive than the previous. Further, each level precisely captures programs that can be proved using uniform Ashcroft invariants with k universal quantifiers. We demonstrate the usefulness of the hierarchy by giving a compositional proof of the MACH shootdown algorithm for TLB consistency. We show a proof at level 2 that shows the algorithm is correct for an arbitrary number of CPUs. However, there is no proof for the algorithm at level 1 which does not involve auxiliary state. http://dl.acm.org/authorize?N29943

On verifying causal consistency

Ahmed Bouajjani, Constantin Enea, Rachid Guerraoui, Jad Hamza

Causal consistency is one of the most adopted consistency criteria for distributed implementations of data structures. It ensures that operations are executed at all sites according to their causal precedence. We address the issue of verifying automatically whether the executions of an implementation of a data structure are causally consistent. We consider two problems: (1) checking whether one single execution is causally consistent, which is relevant for developing testing and bug finding algorithms, and (2) verifying whether all the executions of an implementation are causally consistent. We show that the first problem is NP-complete. This holds even for the read-write memory abstraction, which is a building block of many modern distributed systems. Indeed, such systems often store data in key-value stores, which are instances of the readwrite memory abstraction. Moreover, we prove that, surprisingly, the second problem is undecidable, and again this holds even for the read-write memory abstraction. However, we show that for the read-write memory abstraction, these negative results can be circumvented if the implementations are data independent, i.e., their behaviors do not depend on the data values that are written or read at each moment, which is a realistic assumption. We prove that for data independent implementations, the problem of checking the correctness of a single execution w.r.t. the read-write memory abstraction is polynomial time. Furthermore, we show that for such implementations the set of non-causally consistent executions can be represented by means of a finite number of register automata. Using these machines as observers (in parallel with the implementation) allows to reduce polynomially the problem of checking causal consistency to a state reachability problem. This reduction holds regardless of the class of programs used for the implementation, of the number of read-write variables, and of the used data domain. It allows leveraging existing techniques for assertion/reachability checking to causal consistency verification. Moreover, for a significant class of implementations, we derive from this reduction the decidability of verifying causal consistency w.r.t. the read-write memory abstraction. http://dl.acm.org/authorize?N29054

String Analysis

What is decidable about string constraints with the ReplaceAll function

Taolue Chen, Yan Chen, Matthew Hague, Anthony W. Lin, Zhilin Wu

The theory of strings with concatenation has been widely argued as the basis of constraint solving for verifying string-manipulating programs. However, this theory is far from adequate for expressing many string constraints that are also needed in practice; for example, the use of regular constraints (pattern matching against a regular expression), and the string-replace function (replacing either the first occurrence or all occurrences of a ``pattern'' string constant/variable/regular expression by a ``replacement'' string constant/variable), among many others. Both regular constraints and the string-replace function are crucial for such applications as analysis of JavaScript (or more generally HTML5 applications) against cross-site scripting (XSS) vulnerabilities, which motivates us to consider a richer class of string constraints. The importance of the string-replace function (especially the replace-all facility) is increasingly recognised, which can be witnessed by the incorporation of the function in the input languages of several string constraint solvers.

Recently, it was shown that any theory of strings containing the string-replace function (even the most restricted version where pattern/replacement strings are both constant strings) becomes undecidable if we do not impose some kind of straight-line (aka acyclicity) restriction on the formulas. Despite this, the straight-line restriction is still practically sensible since this condition is typically met by string constraints that are generated by symbolic execution. In this paper, we provide the first systematic study of straight-line string constraints with the string-replace function and the regular constraints as the basic operations. We show that a large class of such constraints (i.e. when only a constant string or a regular expression is permitted in the pattern) is decidable. We note that the string-replace function, even under this restriction, is sufficiently powerful for expressing the concatenation operator and much more (e.g. extensions of regular expressions with string variables). This gives us the most expressive decidable logic containing concatenation, replace, and regular constraints under the same umbrella. Our decision procedure for the straight-line fragment follows an automata-theoretic approach, and is modular in the sense that the string-replace terms are removed one by one to generate more and more regular constraints, which can then be discharged by the state-of-the-art string constraint solvers. We also show that this fragment is, in a way, a maximal decidable subclass of the straight-line fragment with string-replace and regular constraints. To this end, we show undecidability results for the following two extensions: (1) variables are permitted in the pattern parameter of the replace function, (2) length constraints are permitted. https://doi.org/10.1145/3158091

String constraints with concatenation and transducers solved efficiently

Lukáš Holík, Petr Janků, Anthony W. Lin, Philipp Rümmer, Tomáš Vojnar

String analysis is the problem of reasoning about how strings are manipulated by a program. It has numerous applications including automatic detection of cross-site scripting, and automatic test-case generation. A popular string analysis technique includes symbolic executions, which at their core use constraint solvers over the string domain, a.k.a. string solvers. Such solvers typically reason about constraints expressed in theories over strings with the concatenation operator as an atomic constraint. In recent years, researchers started to recognise the importance of incorporating the replace-all operator (i.e. replace all occurrences of a string by another string) and, more generally, finite-state transductions in the theories of

strings with concatenation. Such string operations are typically crucial for reasoning about XSS vulnerabilities in web applications, especially for modelling sanitisation functions and implicit browser transductions (e.g. innerHTML). Although this results in an undecidable theory in general, it was recently shown that the straight-line fragment of the theory is decidable, and is sufficiently expressive in practice. In this paper, we provide the first string solver that can reason about constraints involving both concatenation and finite-state transductions. Moreover, it has a completeness and termination guarantee for several important fragments (e.g. straight-line fragment). The main challenge addressed in the paper is the prohibitive worst-case complexity of the theory (double-exponential time), which is exponentially harder than the case without finite-state transductions. To this end, we propose a method that exploits succinct alternating finite-state automata as concise symbolic representations of string constraints. In contrast to previous approaches using nondeterministic automata, alternation offers not only exponential savings in space when representing Boolean combinations of transducers, but also a possibility of succinct representation of otherwise costly combinations of transducers and concatenation. Reasoning about the emptiness of the AFA language requires a state-space exploration in an exponential-sized graph, for which we use model checking algorithms (e.g. IC3). We have implemented our algorithm and demonstrated its efficacy on benchmarks that are derived from cross-site scripting analysis and other examples in the literature.
https://doi.org/10.1145/3158092

Model Checking and Automata

Learning to Complement Büchi Automata

Yong Li, Andrea Turrini, Lijun Zhang, Sven Schewe

Complementing Büchi automata is an intriguing and intensively studied problem. Complementation suffers from a theoretical super-exponential complexity. From an applied point of view, however, there is no reason to assume that the target language is more complex than the source language. The chance that the smallest representation of a complement language is (much) smaller or (much) larger than the representation of its source should be the same; after all, complementing twice is an empty operation. With this insight, we study the use of learning for complementation. We use a recent learning approach for FDFAs, families of DFAs, that can be used to represent \(\omega \)-regular languages, as a basis for our complementation technique. As a surprising result, it has proven beneficial not to learn an FDFA that represents the complement language of a Büchi automaton (or the language itself, as complementing FDFAs is cheap), but to use it as an intermediate construction in the learning cycle. While the FDFA is refined in every step, the target is an associated Büchi automaton that underestimates the language of a conjecture FDFA. We have implemented our approach and compared it on benchmarks against the algorithms provided in GOAL. The complement automata we produce for large Büchi automata are generally smaller, which makes them more valuable for applications like model checking. Our approach has also been faster in 98% of the cases. Finally we compare the advantages we gain by the novel techniques with advantages provided by the high level optimisations implemented in the state-of-the-art tool SPOT.
https://doi.org/10.1007/978-3-319-73721-8_15

A fixpoint calculus for local and global program flows

Rajeev Alur, Swarat Chaudhuri, P. Madhusudan

We define a new fixpoint modal logic, the visibly pushdown μ-calculus (VP-μ), as an extension of the modal μ-calculus. The models of this logic are execution trees of structured programs where the procedure calls and returns are made visible. This new logic can express pushdown specifications on the model that its classical counterpart cannot, and is motivated by recent work on visibly pushdown languages [4]. We show that our logic naturally captures several interesting program specifications in program verification and dataflow analysis. This includes a variety of program specifications such as computing combinations of local and global program flows, pre/post conditions of procedures, security properties involving the context stack, and interprocedural dataflow analysis properties. The logic can capture flow-sensitive and inter-procedural analysis, and it has constructs that allow skipping procedure calls so that local flows in a procedure can also be tracked. The logic generalizes the semantics of the modal μ-calculus by considering summaries instead of nodes as first-class objects, with appropriate constructs for concatenating summaries, and naturally captures the way in which pushdown models are model-checked. The main result of the paper is that the model-checking problem for VP-μ is effectively solvable against pushdown models with no more effort than that required for weaker logics such as CTL. We also investigate the expressive power of the logic VP-μ: we show that it encompasses all properties expressed by a corresponding pushdown temporal logic on linear structures (caret [2]) as well as by the classical μ-calculus. This makes VP-μ the most expressive known program logic for which algorithmic software model checking is feasible. In fact, the decidability of most known program logics (μ-calculus, temporal logics LTL and CTL, caret, etc.) can be understood by their interpretation in the monadic second-order logic over trees. This is not true for the logic VP-μ, making it a new powerful tractable program logic.
https://doi.org/10.1145/1111037.1111051

LTL Model-Checking for Malware Detection

Fu Song, Tayssir Touili

Nowadays, malware has become a critical security threat. Traditional anti-viruses such as signature-based techniques and code emulation become insufficient and easy to get around. Thus, it is important to have efficient and robust malware detectors. In [20,19], CTL model-checking for PushDown Systems (PDSs) was shown to be a robust technique for malware detection. However, the approach of [20,19] lacks precision and runs out of memory in several cases. In this work, we show that several malware specifications could be expressed in a more precise manner using LTL instead of CTL. Moreover, LTL can express malicious behaviors that cannot be expressed in CTL. Thus, since LTL model-checking for PDSs is polynomial in the size of PDSs while CTL model-checking for PDSs is exponential, we propose to use LTL model-checking for PDSs for malware detection. Our approach consists of: (1) Modeling the binary program as a PDS. This allows to track the program's stack (needed for malware detection). (2) Introducing a new logic (SLTPL) to specify the malicious behaviors. SLTPL is an extension of LTL with variables, quantifiers, and predicates over the stack. (3) Reducing the malware detection problem to SLTPL model-checking for PDSs. We reduce this model checking problem to the emptiness problem of Symbolic Büchi PDSs. We implemented our techniques in a tool, and we applied it to detect several viruses. Our results are encouraging. https://doi.org/10.1007/978-3-642-36742-7_29

Program Logics

A new proof rule for almost-sure termination

Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, Joost-Pieter Katoen

We present a new proof rule for proving almost-sure termination of probabilistic programs, including those that contain demonic non-determinism.
An important question for a probabilistic program is whether the probability mass of all its diverging runs is zero, that is that it terminates "almost surely". Proving that can be hard, and this paper presents a new method for doing so. It applies directly to the program's source code, even if the program contains demonic choice. Like others, we use variant functions (a.k.a. "super-martingales") that are real-valued and decrease randomly on each loop iteration; but our key innovation is that the amount as well as the probability of the decrease are parametric. We prove the soundness of the new rule, indicate where its applicability goes beyond existing rules, and explain its connection to classical results on denumerable (non-demonic) Markov chains.
https://doi.org/10.1145/3158121

Temporal Reasoning about Program Executions

Rajeev Alur

While temporal verification of programs is a topic with a long history, its traditional basis —semantics based on word languages— is ill-suited for modular reasoning about procedural programs. We address this inadequacy by defining the semantics of procedural (potentially recursive) programs in terms of languages of nested words and developing a framework for temporal reasoning around it. This generalization has two benefits. First, this style of reasoning naturally unifies Manna-Pnueli-style temporal reasoning with Hoare-style reasoning about structured programs. Second, it allows verification of "non-regular" properties of specific procedural contexts—for example, "if a procedure acquires a lock, then the same invocation releases it before returning." In this talk, we will first discuss the Nested Word Temporal Logic, a temporal logic for infinite nested words, and argue that it is the "right" logic for temporal reasoning about procedural programs based on theoretical results about decidability of the propositional fragment and first-order expressive-completeness. Then, we will present, sound and relatively complete, proof rules for a variety of classes of properties such as local safety, local response, global safety, and staircase reactivity.
https://doi.org/10.1007/978-3-642-00596-1_2

Vorbesprechung am Freitag 26.01.2018 um 14:15 Uhr im Lichthof 7. Stock