Revamping TVLA: Making Parametric Shape Analysis Competitive
(Tool Paper)

Igor Bogudlov[1], Tal Lev-Ami[1,⋆], Thomas Reps[2,⋆⋆], and Mooly Sagiv[1]

[1] School of Comp. Sci., Tel Aviv Univ.,
{igorbogu,tla,msagiv}@post.tau.ac.il
[2] Comp. Sci. Department, Univ. of Wisconsin, Madison
reps@cs.wisc.edu

Abstract. TVLA is a parametric framework for shape analysis that can be easily instantiated to create different kinds of analyzers for checking properties of programs that use linked data structures. We report on dramatic improvements in TVLA's performance, which make the cost of parametric shape analysis comparable to that of the most efficient specialized shape-analysis tools (which restrict the class of data structures and programs analyzed) without sacrificing TVLA's parametricity. The improvements were obtained by employing well-known techniques from the database community to reduce the cost of extracting information from shape descriptors and performing abstract interpretation of program statements and conditions. Compared to the prior version of TVLA, we obtained as much as 50-fold speedup.

1 Introduction

In this paper, we review recent improvements to TVLA (**T**hree-**V**alued-**L**ogic **A**nalyzer), a system for automatically generating a static-analysis implementation from the operational semantics of a given program [1,2]. In TVLA, a language's small-step structural operational semantics is written in a meta-language based on First-Order Logic with Transitive Closure (FO(TC)). The main idea is that program states are represented as logical structures, and the program's transition system is defined using first-order logical formulas. The abstraction is controlled using a set of *Instrumentation Predicates*, which are defined using FO(TC) formulas and dictate what extra information is tracked for each program state. *Integrity constraints* can be provided in the form of FO(TC) formulas; these express invariant properties of the operational semantics (e.g., each program pointer can point to at most one concrete location).

TVLA is a parametric framework based on the theory of [2]. Given the concrete operational semantics, instrumentation predicates, and integrity constraints, TVLA automatically generates the abstract semantics, and, for each program point, produces a conservative abstract representation of the program states at that point. TVLA is intended as a testbed in which it is easy to try out new ideas for shape abstractions.

⋆ Supported by an Adams Fellowship through the Israel Academy of Sciences and Humanities.
⋆⋆ Supported in part by NSF under grants CCF-0540955 and CCF-0524051.

W. Damm and H. Hermanns (Eds.): CAV 2007, LNCS 4590, pp. 221–225, 2007.

A unique aspect of TVLA is that it automatically generates the abstract transformers from the concrete semantics; these transformers are (i) guaranteed to be sound, and (ii) rather precise—the number of false alarms reported in our applications is very small. The abstract transformers in TVLA are computed in 4 stages:

(i) *Focus*—a partial concretization operation in which each heap cell that will be updated is materialized as a singleton (non-summary) individual, so that it is possible to perform a strong update; (ii) *Update* — in which the update formulas are evaluated using Kleene semantics on the abstract structure to achieve a sound abstract transformer; (iii) *Coerce* — a semantic reduction in which an internal Datalog-style constraint solver uses the instrumentation predicates and integrity constraints to improve the precision of the analysis; and (iv) *Blur*—in which the abstraction function is re-applied (which ensures that the analyis terminates).

Compared to specialized shape-analysis approaches, the above operations incur interpretative overhead. In this paper, we show that using techniques taken from the realm of databases, such as semi-naive evaluation and query optimization [3], TVLA reduces this overhead, and thereby achieves performance comparable to that of state-of-the-art specialized shape analysis without changing TVLA's functionality.

Our technical report [4] contains more details about the implementation and related work; the new version of TVLA is available at [5].

2 Key Improvements

The bottleneck in previous versions of TVLA has been Coerce, which needs to consider interactions among all the predicates. The complexity of Coerce stems from the fact that the number of constraints is linear in the size of the program, and the number of tuples that need to be considered during the evaluation of a constraint is exponential in the number of variables in the constraint.

Coerce translates the definitions of instrumentation predicates and the integrity constraints to Datalog-like constraint rules of the form $\varphi \Rightarrow \psi$, where φ is called the base of the rule and is a conjunction of general formulas, and ψ is called the head of the rule and is a literal. The head and the conjuncts of the base are called atoms. Each atom induces a relation of the tuples that satisfy it. Coerce then applies a constraint by searching for assignments to the free variables of the rule such that the base is known to hold (i.e., evaluates to 1), and the head either does not hold (i.e., evaluates to 0) or may not hold (i.e., evaluates to $\frac{1}{2}$). In the first case it safely discards the structure as inconsistent, and in the second case it attempts to coerce the value to 1 (more details can be found in [2]). This process continues until a fixed point is reached in a way similar to evaluation of Datalog rules in databases [3].

View Maintenance with Semi-Naive Evaluation. Semi-naive evaluation is a well-known technique in database view maintenance to speed up the bottom-up evaluation of Datalog rules [3]. On each iteration of the algorithm, it evaluates the rules in *incremental* fashion; i.e., it only considers variable assignments containing relation tuples that were changed during the previous iteration. All other assignments must have been examined during the previous iteration and thus cannot contribute any new information. Because

the number of changed tuples is usually small compared to the size of the relation, this avoids a considerable amount of computation.

We take this idea one step further by using properties of the TVLA abstract transformer step to avoid fully evaluating the constraints even once: all of the constraints hold before the *Focus* and *Update* steps, and thus the set of violating assignments for the structure is initially empty. Any assignment that violates the constraints must therefore be due to a tuple changed by *Focus* or *Update*. We save the structure before and after the potentially violating step, and calculate the difference. This *delta structure* is used during the first iteration of *Coerce*.

Multi-Constraint Evaluation. TVLA integrity constraints are often symmetric, i.e., give rise to a set of rules, all consisting of the same atoms, such that for each such rule a different atom serves as the (negated) head of the rule and the rest remain in the base.

We introduced the notion of a *multi-constraint* to represent a set of rules that have the same set of atoms. Instead of evaluating these rules one-by-one, we can evaluate them all at once at a cost comparable to that of a single rule evaluation.

A constraint is violated when all of the atoms in the base evaluate to 1 while the negated head evaluates to either 1 or $\frac{1}{2}$. Similarly, a multi-constraint is violated when all of its atoms evaluate to 1, except at most one atom that evaluates to $\frac{1}{2}$ and is the head of some rule.

We evaluate the multi-constraint efficiently by keeping count of the number of $\frac{1}{2}$ values for an assignment while enumerating the relations' tuples.

This technique usually cuts the effective number of constraints in half, and affects the running time of *Coerce* accordingly.

Other Improvements. Many other improvements and techniques were introduced into TVLA, including the following: optimizing the query-evaluation order; precomputing information that only depends on constraint structure, such as atom types and dependencies; tracking modified predicates in each structure for quick constraint filtering; caching and on-demand recomputation of transitive closure; caching of recently-used predicate values and tuple lists for predicate negation.

In addition, we did extensive re-engineering and optimization of the TVLA core geared toward improved performance.

3 Experimental Results

We incorporated the above-mentioned techniques into TVLA. The empirical results from running the new tool on various benchmark examples are presented in Table 1. Table 1 compares the running time of each analysis with both the previously available TVLA version, as well as some specialized shape-analysis tools [6,7].

The benchmark suite consisted of the following examples: singly-linked lists operations, including Merge and Reverse; sorting of linked lists, including insertion sort, bubble sort, and a recursive version of Quicksort (using the extension of [8]); sorted-tree insertion and deletion; analysis of set data structures from [9]; analysis of the Lindstrom scanning algorithm [10,11]; insertion into an AVL tree [12].

For each program, Table 1 uses the following shorthand to indicate the set of properties that the analysis established: CL—cleanness, i.e., absence of memory leaks and

null-pointer dereferences; DI—data-structure invariants were maintained (e.g., treeness in the case of a tree-manipulation program); IS—the output result is isomorphic to the input; TE—termination; SO—the output result is sorted. The column labeled "Structs" indicates the total number of (abstracted) logical structures attached to the program's control-flow graph at the end of the analysis. "N/A" denotes absence of available empirical data for the tool. "B/S" denotes that the analysis is beyond the scope of the tool. The tests were done on a 2.6GHz Pentium PC with 1GB of RAM running XP.[1]

Table 1. Running time comparison results

Program	Properties	Structs	New TVLA	Old TVLA	[6]	[7]
LindstromScan	CL, DI	1285	8.21	63.00	10.85	B/S
LindstromScan	CL, DI, IS, TE	183564	2185.50	18154.00	B/S	B/S
SetRemove	CL, DI, SO	13180	106.20	5152.80	B/S	B/S
SetInsert	CL, DI, SO	299	1.75	22.30	B/S	B/S
DeleteSortedTree	CL, DI	2429	6.14	47.92	4.22	B/S
DeleteSortedTree	CL, DI, SO	30754	104.50	1267.70	B/S	B/S
InsertSortedTree	CL, DI	177	0.85	1.94	0.89	B/S
InsertSortedTree	CL, DI, SO	1103	2.53	12.63	B/S	B/S
InsertAVLTree	CL, DI, SO	1855	27.40	375.60	B/S	B/S
Merge	CL, DI	231	0.95	4.34	0.45	0.15
Reverse	CL, DI	57	0.29	0.45	0.07	0.06
InsertSort	CL, DI	712	3.02	23.53	0.09	0.06
BubbleSort	CL, DI	518	1.70	8.45	0.07	N/A
RecQuickSort	CL, DI	5097	3.92	16.04	B/S	0.30
RecQuickSort	CL, DI, SO	5585	9.22	75.01	B/S	B/S

Table 1 shows that our techniques indeed resulted in considerable speedup vis-a-vis the old version of TVLA: most of the examples run an order of magnitude faster, and in one case a factor of 50 is achieved. Moreover, the new tool's performance is comparable with that of specialized analysis tools, especially on larger examples.

More detailed explanations of the various algorithmic improvements, as well as additional data about the level of improved performance that we obtained, are available in a technical report [4]. The use of semi-naive-evaluation techniques improves the running time of Coerce by a factor of 2, and the total running time by about 50% (Table 2, page 9 of [4]). Various query-optimization techniques (Section 2.4 of [4]) also improve the running time significantly. An additional factor of about 3–4 in the running time comes from extensive engineering changes and code optimization (Section 2.2 of [4]).

References

1. Lev-Ami, T., Sagiv, M.: TVLA: A system for implementing static analyses. In: SAS (2000)
2. Sagiv, M., Reps, T., Wilhelm, R.: Parametric shape analysis via 3-valued logic. In: TOPLAS (2002)
3. Ullman, J.: Database and Knowledge Base Systems, vol. I. W.H. Freeman, New York (1988)

[1] Establishing total correctness of Lindstrom Scan was performed on a 2.4GHz Core 2 Duo with 4GB of RAM. The tests of [7] were done on a 2GHz Pentium Linux PC with 512MB of RAM.

4. Bogudlov, I., Lev-Ami, T., Reps, T., Sagiv, M.: Revamping tvla: Making parametric shape analysis competitive. Technical Report TR-2007-01-01, Tel-Aviv Univ. (2007) http://www.cs.tau.ac.il/~tla/2007/papers/TR-2007-01-01.pdf
5. TVLA system. (Available at http://www.cs.tau.ac.il/~tvla/#DownloadTVLA3)
6. Lev-Ami, T., Immerman, N., Sagiv, M.: Abstraction for shape analysis with fast and precise transformers. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, Springer, Heidelberg (2006)
7. Gotsman, A., Berdine, J., Cook, B.: Interprocedural shape analysis with separated heap abstractions. In: Yi, K. (ed.) SAS 2006. LNCS, vol. 4134, Springer, Heidelberg (2006)
8. Rinetzky, N., Sagiv, M., Yahav, E.: Interprocedural shape analysis for cutpoint-free programs. In: Hankin, C., Siveroni, I. (eds.) SAS 2005. LNCS, vol. 3672, Springer, Heidelberg (2005)
9. Reineke, J.: Shape analysis of sets. Master's thesis, Saarland University (2005)
10. Lindstrom, G.: Scanning list structures without stacks or tag bits. IPL vol. 2(2) (1973)
11. Loginov, A., Reps, T., Sagiv, M.: Automatic verification of the Deutsch-Schorr-Waite tree-traversal algorithm. In: Yi, K. (ed.) SAS 2006. LNCS, vol. 4134, Springer, Heidelberg (2006)
12. Parduhn, S.: Algorithm animation using shape analysis with special regard to binary trees. Master's thesis, Saarland University (2005)