Full text available at: http://dx.doi.org/10.1561/2500000037

Shape Analysis

Full text available at: http://dx.doi.org/10.1561/2500000037

Other titles in Foundations and Trends® in Programming Languages

Progress of Concurrent Objects
Hongjin Liang and Xinyu Feng
ISBN: 978-1-68083-672-1

QED at Large: A Survey of Engineering of Formally Verified Software
Talia Ringer, Karl Palmskog, Ilya Sergey, Milos Gligoric and Zachary Tatlock
ISBN: 978-1-68083-594-6

Reconciling Abstraction with High Performance: A MetaOCaml approach
Oleg Kiselyov
ISBN: 978-1-68083-436-9

Full text available at: http://dx.doi.org/10.1561/2500000037

Shape Analysis

Bor-Yuh Evan Chang
University of Colorado, USA
evan.chang@colorado.edu

Cezara Drăgoi
INRIA, France
and CNRS, PSL University, France
cezarad@di.ens.fr

Roman Manevich
Ben-Gurion University of the Negev
Israel
romanm@cs.bgu.ac.il

Noam Rinetzky
Tel Aviv University
Israel
maon@cs.tau.ac.il

Xavier Rival
INRIA, France
and CNRS, PSL University, France
rival@di.ens.fr

now
the essence of knowledge
Boston — Delft

Full text available at: http://dx.doi.org/10.1561/2500000037

Foundations and Trends® in Programming Languages

Published, sold and distributed by:
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

B.-Y. E. Chang, C. Dragoi, R. Manevich, N. Rinetzky and X. Rival. *Shape Analysis*. Foundations and Trends® in Programming Languages, vol. 6, no. 1–2, pp. 1–158, 2020.

ISBN: 978-1-68083-733-9
© 2020 B.-Y. E. Chang, C. Dragoi, R. Manevich, N. Rinetzky and X. Rival

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, mechanical, photocopying, recording or otherwise, without prior written permission of the publishers.

Photocopying. In the USA: This journal is registered at the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923. Authorization to photocopy items for internal or personal use, or the internal or personal use of specific clients, is granted by now Publishers Inc for users registered with the Copyright Clearance Center (CCC). The 'services' for users can be found on the internet at: www.copyright.com

For those organizations that have been granted a photocopy license, a separate system of payment has been arranged. Authorization does not extend to other kinds of copying, such as that for general distribution, for advertising or promotional purposes, for creating new collective works, or for resale. In the rest of the world: Permission to photocopy must be obtained from the copyright owner. Please apply to now Publishers Inc., PO Box 1024, Hanover, MA 02339, USA; Tel. +1 781 871 0245; www.nowpublishers.com; sales@nowpublishers.com

now Publishers Inc. has an exclusive license to publish this material worldwide. Permission to use this content must be obtained from the copyright license holder. Please apply to now Publishers, PO Box 179, 2600 AD Delft, The Netherlands, www.nowpublishers.com; e-mail: sales@nowpublishers.com

Full text available at: http://dx.doi.org/10.1561/2500000037

Foundations and Trends® in Programming Languages
Volume 6, Issue 1–2, 2020

Editorial Board

Editor-in-Chief

Rupak Majumdari
Max Planck Institute for Software Systems

Editors

Martín Abadi
Google and UC Santa Cruz

Anindya Banerjee
IMDEA Software Instituet

Patrick Cousot
ENS, Paris and NYU

Oege De Moor
University of Oxford

Matthias Felleisen
Northeastern University

John Field
Google

Cormac Flanagan
UC Santa Cruz

Philippa Gardner
Imperial College

Andrew Gordon
Microsoft Research and University of Edinburgh

Dan Grossman
University of Washington

Robert Harper
CMU

Tim Harris
Amazon

Fritz Henglein
University of Copenhagen

Rupak Majumdar
MPI and UCLA

Kenneth McMillan
Microsoft Research

J. Eliot B. Moss
University of Massachusetts, Amherst

Andrew C. Myers
Cornell University

Hanne Riis Nielson
Technical University of Denmark

Peter O'Hearni
University College London

Benjamin C. Pierce
University of Pennsylvania

Andrew Pittsi
University of Cambridge

Ganesan Ramalingami
Microsoft Research

Mooly Sagiv
Tel Aviv University

Davide Sangiorgi
University of Bologna

David Schmidt
Kansas State University

Peter Sewell
University of Cambridge

Scott Stoller
Stony Brook University

Peter Stuckey
University of Melbourne

Jan Vitek
Northeastern University

Philip Wadler
University of Edinburgh

David Walker
Princeton University

Stephanie Weiric
University of Pennsylvania

Full text available at: http://dx.doi.org/10.1561/2500000037

Editorial Scope

Topics

Foundations and Trends® in Programming Languages publishes survey and tutorial articles in the following topics:

- Abstract Interpretation
- Compilation and Interpretation Techniques
- Domain Specific Languages
- Formal Semantics, including Lambda Calculi, Process Calculi, and Process Algebra
- Language Paradigms
- Mechanical Proof Checking
- Memory Management
- Partial Evaluation
- Program Logic
- Programming Language Implementation
- Programming Language Security
- Programming Languages for Concurrency
- Programming Languages for Parallelism
- Program Synthesis
- Program Transformations and Optimizations
- Program Verification
- Runtime Techniques for Programming Languages
- Software Model Checking
- Static and Dynamic Program Analysis
- Type Theory and Type Systems

Information for Librarians

Foundations and Trends® in Programming Languages, 2020, Volume 6, 4 issues. ISSN paper version 2325-1107. ISSN online version 2325-1131. Also available as a combined paper and online subscription.

Full text available at: http://dx.doi.org/10.1561/2500000037

Contents

1 Introduction **4**
1.1 Verifying Pointer-Manipulating Programs 4
1.2 Pointer Analysis 7
1.3 Limitations of Pointer Analyses and Need for More Expressive Abstractions 7
1.4 Shape Analysis 8
1.5 Summary and Survey Outline 11

2 Shape Analysis in a Nutshell **13**
2.1 Running Example 13
2.2 Shape Abstraction for Lists 16
2.3 Shape Analysis for Lists 20
2.4 Instances of Shape Analyses 26
2.5 Summary: The Essence of Shape Analysis 29

3 Generic Shape Analysis **30**
3.1 Programs and Semantics 30
3.2 Shape Abstraction 33
3.3 Abstract Interpretation 34
3.4 Summary 38

Full text available at: http://dx.doi.org/10.1561/2500000037

4 Memory Layout Abstractions **40**
4.1 Dividing Lines . . . 40
4.2 Graph-Based Shape Abstractions . . . 44
4.3 Three-Valued Logic Shape Abstraction . . . 53
4.4 Separation Logic-Based Shape Abstraction . . . 87
4.5 Automata-Based Shape Abstractions . . . 110

5 Extension of Shape Abstractions **119**
5.1 Abstraction of Values Stored into Dynamic Structures . . . 120
5.2 Abstraction of Low-Level Memory Models . . . 125
5.3 Interprocedural Shape Analysis . . . 130

6 Abstractions Exploiting Shape Analysis Principles **136**
6.1 Abstraction of Arrays . . . 136
6.2 Abstraction of Dictionary Structures . . . 141

7 Conclusion **144**

References **148**

Full text available at: http://dx.doi.org/10.1561/2500000037

Shape Analysis

Bor-Yuh Evan Chang[1], Cezara Drăgoi[2], Roman Manevich[3], Noam Rinetzky[4] and Xavier Rival[5]

[1] *University of Colorado, USA; evan.chang@colorado.edu*
[2] *INRIA, France and CNRS, PSL University, France; cezarad@di.ens.fr*
[3] *Ben-Gurion University of the Negev, Israel; romanm@cs.bgu.ac.il*
[4] *Tel Aviv University, Israel; maon@cs.tau.ac.il*
[5] *INRIA, France and CNRS, PSL University, France; rival@di.ens.fr*

ABSTRACT

The computation of semantic information about the behavior of pointer-manipulating programs has been a long standing issue, attacked with diverse and numerous techniques and tools for over 50 years. As usual in automatic verification of infinite-state programs, properties of interest are not computable. Thus, static analyses can only be conservative, leading different analyses to make different tradeoffs between the intricacies of the properties they detect, the precision of their inference procedure and analysis, and the scalability of the analysis.

In this context, *shape analyses* focus on inferring highly complex properties of heap-manipulating programs. These programs utilize data structures which are implemented using an unbounded number of dynamically- (heap-) allocated memory cells interconnected via mutable pointer-links. Because shape analyses have to reason about data structures whose size is not bounded by a fixed, known value, they cannot track explicitly the particular properties of every concrete memory cell which the program uses, as done,

Bor-Yuh Evan Chang, Cezara Drăgoi, Roman Manevich, Noam Rinetzky and Xavier Rival (2020), "Shape Analysis", Foundations and Trends® in Programming Languages: Vol. 6, No. 1–2, pp 1–158. DOI: 10.1561/2500000037.

Full text available at: http://dx.doi.org/10.1561/2500000037

e.g., by analysis of variable-manipulating non-recursive programs. Instead, shape analyses *summarize* memory regions by letting one piece of abstract information, called *summary predicate*, describe several concrete cells. The need to cope with data structures of unbounded sizes is a challenge shape analyses share with static analyzers of array-manipulating programs. However, while the size of an array may change in different executions, its layout (i.e., its dimensions and the way its contents are spread over the memory) is fixed. In contrast, the layout of a pointer-linked data structure, colloquially referred to as its *shape*, may evolve dynamically during the program execution and a memory cell can be part of different data structures at different points in time. As a result, shape analyses need to let the denotation of summary predicates in terms of the constituents and layouts of the memory regions which they represent evolve during the analysis as well.

In this survey, we consider that shape analyses are characterized and defined by the presence of summary predicates describing a set of concrete memory cells that varies during the course of the analysis. We use this characterization as a means for distinguishing shape analyses as a particular class of pointer analyses. We show that many "standard" pointer analyses do not fit the aforementioned description, while many analyses relying on very different mathematical foundations, e.g., shape graphs, three-valued logic, and separation logic, do.

The ambition of this survey is to provide a comprehensive introduction to the field of shape analysis, and to present the foundation of this topic, in a single document that is accessible to readers who are not familiar with it. To do so, we characterize the essence of shape analysis compared to more classical pointer analyses. We supply the intuition underlying the abstractions commonly used in shape analysis and the algorithms that allow to statically compute intricate

Full text available at: http://dx.doi.org/10.1561/2500000037

semantic properties. Then, we cover the main families of shape analysis abstraction and algorithms, highlight the similarities between them, and also characterize the main differences between the most common approaches. Last, we review a few other static analysis works (such as array abstractions, dictionary abstractions and interprocedural analyses) that were influenced by the ideas of shape analysis, so as to demonstrate the impact of the field.

Full text available at: http://dx.doi.org/10.1561/2500000037

1

Introduction

1.1 Verifying Pointer-Manipulating Programs

Pointers and dynamic memory allocation are present in one form or another in many modern programming languages and significantly contribute to their expressiveness. For instance, they enable maintaining mutable data structures such as lists, trees, and graphs. The size of such structures may vary during the execution, as cells can be dynamically allocated in the heap when the program needs them in order to store new data. Moreover, the links between elements may be modified locally without changing the whole structure, e.g., to insert a new element into its proper location inside a sorted list. Similarly, common implementations of functional or object oriented languages also make great use of both pointers and dynamic memory allocation so as to represent the call stack, closures, and objects.

On the other hand, these features make reasoning over programs very difficult since the layout of the memory states heavily depends on the program executions. As a consequence, using such features is a notoriously hard task for programmers, and bugs related to them are both common and challenging to diagnose. Depending on the programming language, pointer manipulation errors may cause abrupt crashes due to

Full text available at: http://dx.doi.org/10.1561/2500000037

runtime errors (as the dereference of a null pointer), memory leakage, i.e., make memory blocks unreachable, and thus impossible to ever deallocate, cause pointers to become dangling, i.e., point to (manually) deallocated memory regions, which may lead to further pointer related errors, e.g., memory corruptions (a write through a dangling pointer, that happens to refer to a memory area that has been freed and then allocated again to store other, unrelated, data).

On top of that, the preservation of structural invariants of pointer-linked data structures is often non-trivial, as a pointer manipulation error might create a cycle in a structure that is supposed to be acyclic and/or leak a large part of it. As an example, Figure 1.1 displays several common examples of dynamic data structures, with very different properties:

- *singly-linked lists* consist of acyclic chains of elements ending with a special element, and where the link from one element to the next usually boils down to a pointer field embedded in every element;
- *doubly-linked lists* augment the singly-linked list structure with backward pointers from each element to its predecessor;
- *circular lists* have the same local structure as the singly-linked lists, but form a loop, so that it is always possible to access the successor of any element;
- *binary trees* are also chained structures, but are such that each non-leaf node has a left and a right successor (a slightly different definition of binary trees accepts structures where some nodes may have no left child or no right child);
- *binary trees with parent pointers* augment binary trees with backward links from every node to its predecessor, Similarly to the way doubly-linked lists augments singly-linked lists with back-pointers;
- *connected graphs* consist of sets of elements, such that each element has a number of successors who are also elements of the structure; in particular, they may contain cycles, elements with no successors, etc.

Full text available at: http://dx.doi.org/10.1561/2500000037

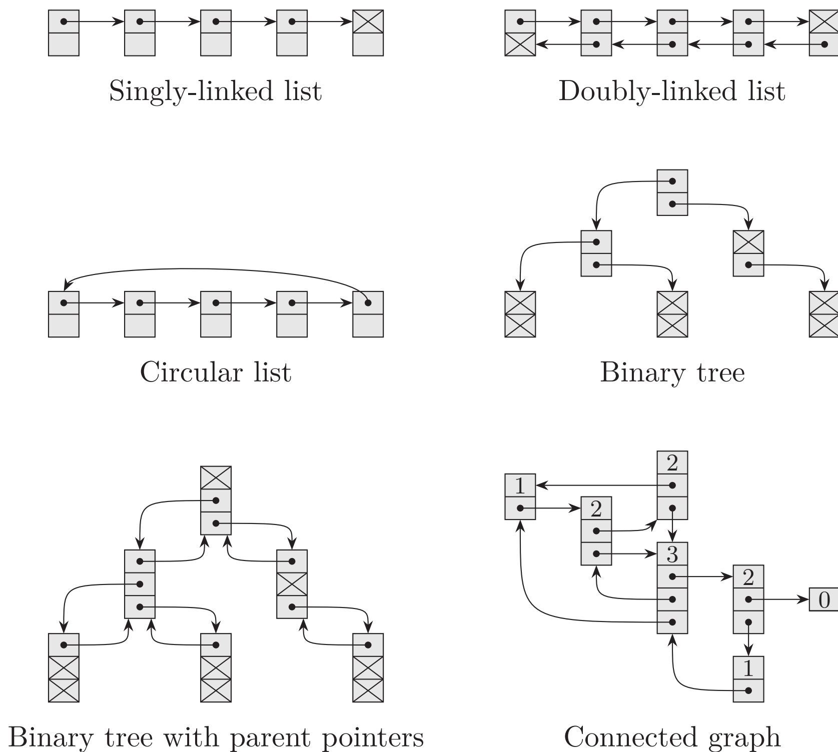

Figure 1.1: A few unbounded and dynamic data structures.

This defines just a small sample of the structures one can imagine, and it is possible to combine these patterns or invent others, e.g., a list of trees or a tree the nodes of which are also connected by a list. Each structure comes with a set of properties (existence of chains of links to next elements, reachability, absence or existence of cycles, existence of a linear order or not…). Furthermore, the correct utilization of each structure relies on the preservation of its *shape invariant*—a combination of global properties pertaining to the layout of its elements—which is generally hard to establish.

Due to these difficulties, a large number of works have searched for techniques to reason about pointer-manipulating programs automatically so as to verify the aforementioned properties. In general, *static*

Full text available at: http://dx.doi.org/10.1561/2500000037

analysis aims at computing automatically semantic properties of programs, namely properties that are satisfied by every program execution, such as the absence of some classes of errors, or the preservation of some invariants. Broadly speaking, there are two (somewhat overlapping) categories of static analysis of heap-manipulating pointer programs: *pointer analyses* and *shape analyses*, as we discuss next.

1.2 Pointer Analysis

Pointer analyses (see Smaragdakis and Balatsouras, 2015 for a recent survey) attempt to determine properties of pointer values and of the structures they refer to. A first useful property is the validity of pointer values, which expresses that they are neither dangling nor null. While it is useful in order to prove that some errors such as a null/dangling pointer dereference or the corruption of an unknown memory location cannot occur, this property is often too weak to fully understand what a program does. A second useful semantic property focuses on the resolution of pointers so as to determine to which address a pointer may refer, or what pairs of pointers may be equal (alias). This property is extremely useful to resolve memory accesses, and help basically any kind of program reasoning technique when considering a program that manipulates pointers. *Points-to analyses* such as Andersen (1994) or Steensgaard (1996) compute a super-set of the addresses each pointer variable may refer to. Essentially, each memory cell with a pointer type is mapped into a set of symbolic addresses it may point to, and this set can be used so as to resolve memory accesses. *Alias analyses* such as Cooper and Kennedy (1989) compute a super-set of the aliasing relation between pointers, which is another way to describe the topology of pointers (see, e.g., Jonkers and Jonkers, 1981).

1.3 Limitations of Pointer Analyses and Need for More Expressive Abstractions

Points-to and alias analyses rely on basic and generally cheap abstractions of program states, and can often be carried out in a fully flow-insensitive manner for better performance, relying on field-, object

Full text available at: http://dx.doi.org/10.1561/2500000037

creation site, or context-sensitivity to improve precision. On the other hand, the range of properties they may infer is typically quite limited. In general, when the size of data structures or the numbers of allocated memory blocks are unbounded, many important properties fall beyond the scope of these analyses. As an example, the *reachability* of a cell that is allocated dynamically becomes hard to establish since the chains of pointers from program variables to it may be arbitrarily long. This property is important in order to verify the absence of memory leaks in languages where deallocation is manual. Similarly, the *acyclicity* of a data structure expresses the absence of certain patterns in pointer paths, can only be established by reasoning over arbitrarily long paths. This property is important in order to verify structural preservation or termination of loops. The key issue is that these properties are not local, and can only be justified by global arguments. In fact, it is not rare that even the verification of a local property, e.g., pointer validity, requires establishing a global property, e.g., reachability.

There exist techniques to make pointer analyses less local and extend their expressiveness. As an example, Deutsch (1994) infers aliasing relations over access paths that are of unbounded length, and that can be tied together by the means of numeric relations: this analysis can express that some pointer stores the address of an element that lies somewhere in the middle of a list-like structure. However, such techniques remain limited, and cannot express that a list (or an instance of some other dynamic structure) is well-formed.

1.4 Shape Analysis

Shape analyses, in contrast to pointer analyses, aim at computing global structural properties of unbounded sets of memory cells and pointers, such as the shape invariants of the data structures depicted in Figure 1.1. An example of shape property is the well-formedness of a singly linked list or that of a binary tree without sharing. Such properties concern an unbounded number of memory cells, and tightly constrain correlations between an unbounded number of pointers fields. This allows them to convey, for instance, the absence of cycles over arbitrarily long link

Full text available at: http://dx.doi.org/10.1561/2500000037

chains. Such relations are intrinsically harder to define and reason about than relations over finite sets of pointers or of regions.

Shape analyses have in common a much higher level of expressiveness than the aforementioned pointer analysis and they rely on very different basic logical predicates. In particular, each of them features some kinds of basic predicates that are able to *summarize* memory regions of unbounded size and in a compact manner while retaining some global information about the shape properties of the summarized region. This is absolutely required to express shape properties over unbounded data structures such as lists, trees and graphs: indeed, abstractions that lack the ability to summarize are either limited to keeping precision on finite sets of memory cells, while losing precision on the rest, or require to resort to a possibly unbounded number of disjuncts.

In addition to summarization, shape analyses need to calculate precisely how program statements transform summaries. In practice, they often need to temporarily refine summaries in order to reason precisely over program statements that impact them. This process, often called *materialization* or *focus*, allows the analysis to apply case analysis regarding the layout of the heap part represented by a summary predicate. Materialization allows to perform strong updates of heap cells located deep in the heap as it enables the analysis to dynamically refine its view of the parts of the heap that pointer variables refer to when analyzing, e.g., the traversal of unbounded data structures.

The use of materialization implies that the analysis also needs to be able to introduce summaries by a generalization process, from more precise predicates. As a consequence, the analysis needs to go back and forth between its base view of data structures and a more refined one, that makes reasoning over local read and destructive update (field mutation) operations possible.

Materializing and Non-Materializing Shape Analyses. In the following, we distinguish between two families of shape analyses: the first category is unable to do materialization at any time and thus can perform strong updates only when certain favorable conditions hold, and the second category that is able to perform dynamic materialization

Full text available at: http://dx.doi.org/10.1561/2500000037

(at any time during the analysis) and thus is able to perform strong updates in more cases.

Non-Materializing Shape Analyses. As an example for the latter kind of analyses, Ghiya and Hendren (1996) uses global predicates that state that some structures are "tree like", that is, acyclic and without sharing, or simply "DAG like", that is, acyclic, but possibly with some amount of internal sharing. Unlike the pointer analyses mentioned above, this analysis actually captures properties related to the shape of heap data structures that are manipulated by programs.

Materializing Shape Analyses. Two notable examples for the kind of shape analyses which use materialization are the three-valued logic framework for shape analysis of Sagiv *et al.* (1999, 2002), and analyses based on separation logic which was introduced by Reynolds (2002) and Ishtiaq and O'Hearn (2001).

Three-valued logic relies on basic user-defined shape predicates (such as local points-to predicates, global reachability predicates expresses by transitive closure over the points-to predicates, and acyclicity predicates) and summary nodes that stand for unbounded numbers of concrete memory cells or addresses in order to describe large families of shape properties of heap data structures. TVLA (Lev-Ami and Sagiv, 2000) is a parametric system which can very precisely capture structures such as lists or graphs, and it was applied to a wide range of shape analysis problems.

Separation logic was proposed as a language to tie logical properties to heap regions. As an example, it can naturally convey, thanks to the so-called *separating conjunction*, that a memory region can be divided into a finite set of pairwise disjoint regions that store specific data structures, and that can be reasoned about in a separate manner. This is the basis of *local reasoning*, which simplifies the analysis of atomic program statements by letting it focus on the memory cells that they may read or update. Coupled with inductive predicates, separation logic can describe many interesting data structures of unbounded size, and assert that a region stores, e.g., a well-formed singly linked list

Full text available at: http://dx.doi.org/10.1561/2500000037

or a well-formed binary tree with no sharing. It has served as a basis for several static analyses including those described in Distefano *et al.* (2006), Berdine *et al.* (2007), Chang *et al.* (2007), Dudka *et al.* (2011), or Holík *et al.* (2013).

Applications of Shape Analysis. Besides memory safety and the verification of correctness properties for sequential programs as outlined above, we can cite many applications for shape analysis techniques. An important example is the case of parallel programs, where several threads may concurrently access and modify shared data-structures. Among the many works that have attacked this problem, we can cite Berdine *et al.* (2008), Manevich *et al.* (2008), and Vafeiadis (2010). In general, the works rely on shape abstractions that are rather similar to those used in the sequential case and compute information about the thread interaction in terms of heap abstraction.

More surprisingly, shape analysis abstraction also have applications far outside the world of program analysis. For instance, Srivastava *et al.* (2011) reduces the search of solutions for planning problems to shape analysis problems.

1.5 Summary and Survey Outline

The goal of this survey is to survey the main shape analysis techniques and to convey a general understanding of the main characteristics of these static analyses. As it is not possible to provide an exhaustive recollection of all the works carried out on this topic, we adopt a more modest approach and focus on the main principles related to abstraction (namely, the relation between concrete stores and abstract predicates), to the computation of post-conditions for atomic operations and to the generalization of abstract predicates to enforce termination of analyses. In this process, we intend to highlight similarities and differences among the main approaches. Moreover, the principles underlying shape analysis also inspired other static analyses aimed at programs manipulating other classes of data structures such as arrays or dictionaries. Thus, we also show the link between shape analyses and other families of abstractions and static analysis.

Full text available at: http://dx.doi.org/10.1561/2500000037

This survey has the following structure. Section 2 presents an intuitive overview of the main principles of shape analysis, without adopting one specific formalism. In fact, it mostly only relies on a graphical presentation. Section 3 formalizes a concrete model of program states and executions to be used in the rest of the survey. As often, the choice of the concrete model of programs deeply influences the ensuing definition of abstractions and static analysis algorithms. Section 4 integrates some of the main approaches to shape analysis into this framework. This is the core part of this survey, since it defines and formalizes the main abstractions and analysis algorithms. Section 5 presents important extensions of shape analysis, so as to describe not only the shape of memory, but also the content and the low level layout of data structures and to analyze programs with functions and procedures. Section 6 describes a few abstractions and static analyses that rely on principles that are similar to the main foundational techniques of shape analysis abstractions and algorithms. Finally, Section 7 draws the main conclusions of our study.

Full text available at: http://dx.doi.org/10.1561/2500000037

References

Abdulla, P. A., L. Holík, B. Jonsson, O. Lengál, C. Q. Trinh, and T. Vojnar (2016). "Verification of heap manipulating programs with ordered data by extended forest automata". *Acta Informatica*. 53(4): 357–385.

Andersen, L. O. (1994). "Program analysis and specialization for the C programming language". *PhD thesis*. DIKU, University of Copenhagen.

Balaban, I., A. Pnueli, and L. D. Zuck (2007). "Shape analysis of single-parent heaps". In: *Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*. Ed. by B. Cook and A. Podelski. Vol. 4349. *Lecture Notes in Computer Science*. Springer. 91–105.

Berdine, J., C. Calcagno, and P. W. O'Hearn (2005a). "Smallfoot: Modular automatic assertion checking with separation logic". In: *Formal Methods for Components and Objects, 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, November 1–4, Revised Lectures*. 115–137.

Berdine, J., C. Calcagno, and P. W. O'Hearn (2005b). "Symbolic execution with separation logic". In: *APLAS*. Springer. 52–68.

Berdine, J., C. Calcagno, B. Cook, D. Distefano, P. O'Hearn, T. Wies, and H. Yang (2007). "Shape analysis for composite data structures". In: *International Conference on Computer Aided Verification (CAV)*. Springer. 178–192.

Full text available at: http://dx.doi.org/10.1561/2500000037

Berdine, J., T. Lev-Ami, R. Manevich, G. Ramalingam, and S. Sagiv (2008). "Thread quantification for concurrent shape analysis". In: *International Conference on Computer Aided Verification (CAV)*. Ed. by A. Gupta and S. Malik. Vol. 5123. *Lecture Notes in Computer Science*. Springer. 399–413.

Berdine, J., B. Cook, and S. Ishtiaq (2011). "Slayer: Memory safety for systems-level code". In: *Computer Aided Verification – 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14–20, Proceedings*. 178–183.

Bouajjani, A., M. Bozga, P. Habermehl, R. Iosif, P. Moro, and T. Vojnar (2006). "Programs with lists are counter automata". In: *International Conference on Computer Aided Verification (CAV)*. 517–531.

Bouajjani, A., C. Drăgoi, C. Enea, and M. Sighireanu (2011). "On interprocedural analysis of programs with lists and data". In: *Conference on Programming Languages Design and Implementation (PLDI)*. 578–589.

Bouajjani, A., C. Drăgoi, C. Enea, and M. Sighireanu (2012). "Abstract domains for automated reasoning about list-manipulating programs with infinite data". In: *Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*. 1–22.

Brockschmidt, M., Y. Chen, P. Kohli, S. Krishna, and D. Tarlow (2017). "Learning shape analysis". In: *Static Analysis Symposium (SAS)*. Ed. by F. Ranzato. Vol. 10422. *Lecture Notes in Computer Science*. Springer. 66–87.

Calcagno, C. and D. Distefano (2011). "Infer: An automatic program verifier for memory safety of C programs". In: *NASA Formal Methods – Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18–20, Proceedings*. 459–465.

Calcagno, C., D. Distefano, P. W. O'Hearn, and H. Yang (2006). "Beyond reachability: Shape abstraction in the presence of pointer arithmetic". In: *Static Analysis Symposium (SAS)*. Springer. 182–203.

Calcagno, C., D. Distefano, P. W. O'Hearn, and H. Yang (2007). "Footprint analysis: A shape analysis that discovers preconditions". In: *Static Analysis, 14th International Symposium, SAS 2007, Kongens Lyngby, Denmark, August 22–24, Proceedings*. 402–418.

Full text available at: http://dx.doi.org/10.1561/2500000037

Calcagno, C., D. Distefano, P. W. O'Hearn, and H. Yang (2009). "Compositional shape analysis by means of bi-abduction". In: *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL).* Savannah, GA, USA, January 21–23. 289–300.

Chang, B.-Y. E. and X. Rival (2008). "Relational inductive shape analysis". In: *Symposium on Principles of Programming Languages (POPL)*. ACM. 247–260.

Chang, B.-Y. E. and X. Rival (2013). "Modular construction of shape-numeric analyzers". In: *Semantics, Abstract Interpretation, and Reasoning About Programs: Essays Dedicated to David A. Schmidt on the Occasion of His Sixtieth Birthday, Manhattan, KS, USA, September 19–20*. 161–185.

Chang, B.-Y. E., X. Rival, and G. Necula (2007). "Shape analysis with structural invariant checkers". In: *Static Analysis Symposium (SAS)*. Springer. 384–401.

Chase, D. R., M. Wegman, and F. K. Zadeck (1990). "Analysis of pointers and structures". In: *Proceedings of the ACM SIGPLAN 1990 Conference on Programming Language Design and Implementation. PLDI '90*. New York, USA: ACM. 296–310.

Chin, W.-N., C. David, H. H. Nguyen, and S. Qin (2007). "Automated verification of shape, size and bag properties". In: *12th International Conference on Engineering of Complex Computer Systems (ICECCS 2007)*. 307–320.

Cooper, K. D. and K. Kennedy (1989). "Fast interprocedural alias analysis". In: *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. ACM. 49–59.

Cousot, P. and R. Cousot (1977). "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints". In: *Symposium on Principles of Programming Languages (POPL)*. ACM. 238–252.

Cousot, P. and R. Cousot (1979). "Systematic design of program analysis frameworks". In: *Symposium on Principles of Programming Languages (POPL)*. 269–282.

Full text available at: http://dx.doi.org/10.1561/2500000037

Cousot, P. and N. Halbwachs (1978). “Automatic discovery of linear restraints among variables of a program”. In: *Symposium on Principles of Programming Languages (POPL)*. Tucson, AZ: ACM. 84–97.

Cousot, P., R. Cousot, and F. Logozzo (2011). “A parametric segmentation functor for fully automatic and scalable array content analysis”. In: *POPL*. Austin, TX, USA: ACM. 105–118.

Cox, A., B.-Y. E. Chang, and X. Rival (2014). “Automatic analysis of open objects in dynamic language programs”. In: *Static Analysis Symposium (SAS)*. Springer. 134–150.

Cox, A., B.-Y. E. Chang, and X. Rival (2015). “Desynchronized multi-state abstractions for open programs in dynamic languages”. In: *European Symposium on Programming (ESOP)*. Springer. 483–509.

Deutsch, A. (1994). “Interprocedural may-alias analysis for pointers: Beyond k-limiting”. In: *Proceedings of the Conference on Programming Language Design and Implementation (PLDI)*. ACM. 230–241.

Dillig, I., T. Dillig, and A. Aiken (2011). “Precise reasoning for programs using containers”. In: *Symposium on Principles of Programming Languages (POPL)*. ACM. 187–200.

Distefano, D., P. O’Hearn, and H. Yang (2006). “A local shape analysis based on separation logic”. In: *Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer. 287–302.

Dor, N., J. Field, D. Gopan, T. Lev-Ami, A. Loginov, R. Manevich, G. Ramalingam, T. W. Reps, N. Rinetzky, M. Sagiv, R. Wilhelm, E. Yahav, and G. Yorsh (2005). “Automatic verification of strongly dynamic software systems”. In: *Verified Software: Theories, Tools, Experiments, First IFIP TC 2/WG 2.3 Conference, VSTTE 2005, Zurich, Switzerland, October 10–13, Revised Selected Papers and Discussions*. 82–92.

Dudka, K., P. Peringer, and T. Vojnar (2011). “Predator: A practical tool for checking manipulation of dynamic data structures using separation logic”. In: *International Conference on Computer Aided Verification (CAV)*. Springer. 372–378.

Ghiya, R. and L. J. Hendren (1996). “Is it a tree, a DAG, or a cyclic graph? A shape analysis for heap-directed pointers in C”. In: *Symposium on Principles of Programming Languages (POPL)*. 1–15.

Full text available at: http://dx.doi.org/10.1561/2500000037

Gotsman, A., J. Berdine, and B. Cook (2006). "Interprocedural shape analysis with separated heap abstractions". In: *Static Analysis, 13th International Symposium, SAS 2006, Seoul, Korea, August 29–31, Proceedings*. 240–260.

Graf, S. and H. Saïdi (1997). "Construction of abstract state graphs with PVS". In: *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22–25, Proceedings*. 72–83.

Habermehl, P., R. Iosif, and T. Vojnar (2006). "Automata-based verification of programs with tree updates". In: *Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. 350–364.

Habermehl, P., L. Holík, A. Rogalewicz, J. Simácek, and T. Vojnar (2011). "Forest automata for verification of heap manipulation". In: *International Conference on Computer Aided Verification (CAV)*. Springer. 424–440.

Habermehl, P., L. Holík, A. Rogalewicz, J. Simácek, and T. Vojnar (2012). "Forest automata for verification of heap manipulation". *Formal Methods in System Design (FMSD)*. 41(1): 83–106.

Halbwachs, N. and M. Péron (2008). "Discovering properties about arrays in simple programs". In: *PLDI*. Tucson, AZ, USA: ACM. 339–348.

Holík, L., O. Lengál, A. Rogalewicz, J. Simácek, and T. Vojnar (2013). "Fully automated shape analysis based on forest automata". In: *International Conference on Computer Aided Verification (CAV)*. Springer. 740–755.

Illous, H., M. Lemerre, and X. Rival (2017). "A relational shape abstract domain". In: *NASA Formal Methods – 9th International Symposium, NFM 2017*. Vol. 10227. *Lecture Notes in Computer Science*. Springer. 212–229.

Iosif, R., A. Rogalewicz, and T. Vojnar (2014). "Deciding entailments in inductive separation logic with tree automata". In: *Automated Technology for Verification and Analysis (ATVA)*. Springer. 201–218.

Ishtiaq, S. S. and P. W. O'Hearn (2001). "BI as an assertion language for mutable data structures". In: *Symposium on Principles of Programming Languages (POPL)*. ACM. 14–26.

Full text available at: http://dx.doi.org/10.1561/2500000037

Itzhaky, S., A. Banerjee, N. Immerman, A. Nanevski, and M. Sagiv (2013). "Effectively-propositional reasoning about reachability in linked data structures". In: *International Conference on Computer Aided Verification (CAV)*. Ed. by N. Sharygina and H. Veith. Vol. 8044. *Lecture Notes in Computer Science*. Springer. 756–772.

Jeannet, B., A. Loginov, T. W. Reps, and S. Sagiv (2004). "A relational approach to interprocedural shape analysis". In: *Static Analysis Symposium (SAS)*. 246–264.

Jeannet, B., A. Loginov, T. W. Reps, and M. Sagiv (2010). "A relational approach to interprocedural shape analysis". *ACM Trans. Program. Lang. Syst.* 32(2): 5:1–5:52.

Jones, N. D. and S. S. Muchnick (1979). "Flow analysis and optimization of LISP-like structures". In: *Proceedings of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*. New York, NY, USA: ACM. 244–256. URL: http://doi.acm.org/10.1145/567752.567776. Reprinted in *Program Flow Analysis: Theory and Application*, Muchnick, Steven S. and Jones, Neil D., 1981, published by Prentice Hall Professional Technical Reference.

Jones, N. D. and S. S. Muchnick (1982). "A flexible approach to interprocedural data flow analysis and programs with recursive data structures". In: *Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. New York, NY, USA: ACM. 66–74.

Jonkers, H. B. and H. B. M. Jonkers (1981). *Abstract Storage Structures*. Afdeling Informatica: IW. Afdeling Informatica, Mathematisch Centrum.

Kleene, S. C. (1952). *Introduction to Metamathematics*. Vol. 483. New York, NY: D. Van Nostrand Co., Inc.

Kreiker, J., H. Seidl, and V. Vojdani (2010). "Shape analysis of low-level C with overlapping structures". In: *Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*. Springer. 214–230.

Larus, J. R. and P. N. Hilfinger (1988). "Detecting conflicts between structure accesses". In: *Proceedings of the ACM SIGPLAN 1988 Conference on Programming Language Design and Implementation. PLDI '88*. New York, NY, USA: ACM. 24–31.

Full text available at: http://dx.doi.org/10.1561/2500000037

Laviron, V., B.-Y. E. Chang, and X. Rival (2010). "Separating shape graphs". In: *European Symposium on Programming (ESOP)*. Springer. 387–406.

Le, Q. L., J. Sun, and W.-N. Chin (2016). "Satisfiability modulo heap-based programs". In: *International Conference on Computer Aided Verification (CAV)*. Ed. by S. Chaudhuri and A. Farzan. Vol. 9779. *Lecture Notes in Computer Science*. Springer. 382–404.

Lee, O., H. Yang, and R. Petersen (2011). "Program analysis for overlaid data structures". In: *International Conference on Computer Aided Verification (CAV)*. 592–608.

Lev-Ami, T. and S. Sagiv (2000). "TVLA: A system for implementing static analyses". In: *Static Analysis Symposium (SAS)*. Springer. 280–301.

Li, H., X. Rival, and B.-Y. E. Chang (2015). "Shape analysis for unstructured sharing". In: *Static Analysis Symposium (SAS)*. 90–108.

Li, H., F. Berenger, B.-Y. E. Chang, and X. Rival (2017). "Semantic-directed clumping of disjunctive abstract states". In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*. Paris, France, January 18–20. 32–45.

Liu, J. and X. Rival (2015). "Abstraction of arrays based on non contiguous partitions". In: *Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*. Mumbai, India: Springer. 282–299.

Liu, J., L. Chen, and X. Rival (2018). "Automatic verification of embedded system code manipulating dynamic structures stored in contiguous regions". In: *International Conference on Embedded Software (EMSOFT)*.

Loginov, A., T. W. Reps, and S. Sagiv (2005). "Abstraction refinement via inductive learning". In: *International Conference on Computer Aided Verification (CAV)*. Ed. by K. Etessami and S. K. Rajamani. Vol. 3576. *Lecture Notes in Computer Science*. Springer. 519–533.

Madhusudan, P., G. Parlato, and X. Qiu (2011). "Decidable logics combining heap structures and data". In: *Symposium on Principles of Programming Languages (POPL)*. Ed. by T. Ball and M. Sagiv. ACM. 611–622.

Full text available at: http://dx.doi.org/10.1561/2500000037

Magill, S., J. Berdine, E. M. Clarke, and B. Cook (2007). "Arithmetic strengthening for shape analysis". In: *Static Analysis, 14th International Symposium, SAS 2007, Kongens Lyngby, Denmark, August 22–24, Proceedings*. 419–436.

Magill, S., M.-H. Tsai, P. Lee, and Y.-K. Tsay (2010). "Automatic numeric abstractions for heap-manipulating programs". In: *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. Madrid, Spain, January 17–23. 211–222.

Manevich, R., T. Lev-Ami, M. Sagiv, G. Ramalingam, and J. Berdine (2008). "Heap decomposition for concurrent shape analysis". In: *Static Analysis Symposium (SAS)*. Ed. by M. Alpuente and G. Vidal. Vol. 5079. *Lecture Notes in Computer Science*. Springer. 363–377.

Manevich, R., B. Dogadov, and N. Rinetzky (2016). "From shape analysis to termination analysis in linear time". In: *Computer Aided Verification – 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17–23, Proceedings, Part I*. 426–446.

Might, M. (2010). "Shape analysis in the absence of pointers and structure". In: *Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*. Springer. 263–278.

Miné, A. (2006). "Field-sensitive value analysis of embedded C programs with union types and pointer arithmetics". In: *Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*. ACM. 54–63.

Nelson, G. (1983). "Verifying reachability invariants of linked structures". In: *Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*. New York, NY, USA: ACM. 38–47.

O'Hearn, P. W. and D. J. Pym (1999). "The logic of bunched implications". *Bulletin of Symbolic Logic*. 5(2): 215–244.

O'Hearn, P. W., J. C. Reynolds, and H. Yang (2001). "Local reasoning about programs that alter data structures". In: *International Conference on Computer Science Logics (CSL)*. 1–19.

Full text available at: http://dx.doi.org/10.1561/2500000037

Pek, E., X. Qiu, and P. Madhusudan (2014). "Natural proofs for data structure manipulation in C using separation logic". In: *Conference on Programming Languages Design and Implementation (PLDI)*. Ed. by M. F. P. O'Boyle and K. Pingali. ACM. 440–451.

Piskac, R., T. Wies, and D. Zufferey (2013). "Automating separation logic using SMT". In: *International Conference on Computer Aided Verification (CAV)*. Ed. by N. Sharygina and H. Veith. Vol. 8044. *Lecture Notes in Computer Science*. Springer. 773–789.

Podelski, A. and T. Wies (2005). "Boolean heaps". In: *Static Analysis Symposium (SAS)*. Ed. by C. Hankin and I. Siveroni. Vol. 3672. *Lecture Notes in Computer Science*. Springer. 268–283.

Qiu, X., P. Garg, A. Stefanescu, and P. Madhusudan (2013). "Natural proofs for structure, data, and separation". In: *Conference on Programming Languages Design and Implementation (PLDI)*. Ed. by H.-J. Boehm and C. Flanagan. ACM. 231–242.

Reps, T. W., S. Horwitz, and M. Sagiv (1995). "Precise interprocedural dataflow analysis via graph reachability". In: *Symposium on Principles of Programming Languages (POPL)*. ACM. 49–61.

Reps, T., M. Sagiv, and A. Loginov (2003). "Finite differencing of logical formulas for static analysis". In: *European Symposium on Programming (ESOP)*. 380–398.

Reynolds, J. C. (1968). "Automatic computation of data set definitions". In: *Information Processing 68: Proceedings of the IFIP Congress*. North-Holland. 296–310.

Reynolds, J. C. (2002). "Separation logic: A logic for shared mutable data structures". In: *Symposium on Logics in Computer Science (LICS)*. IEEE. 55–74.

Rinetzky, N. and S. Sagiv (2001). "Interprocedural shape analysis for recursive programs". In: *International Conference on Compiler Construction (CC)*. Springer. 133–149.

Rinetzky, N., J. Bauer, T. W. Reps, S. Sagiv, and R. Wilhelm (2005a). "A semantics for procedure local heaps and its abstractions". In: *Symposium on Principles of Programming Languages (POPL)*. 296–309.

Full text available at: http://dx.doi.org/10.1561/2500000037

Rinetzky, N., M. Sagiv, and E. Yahav (2005b). "Interprocedural shape analysis for cutpoint-free programs". In: *Static Analysis Symposium (SAS)*. 284–302.

Rival, X. and B.-Y. E. Chang (2011). "Calling context abstraction with shapes". In: *Symposium on Principles of Programming Languages (POPL)*. ACM. 173–186.

Sagiv, M., T. Reps, and R. Wilhelm (1998). "Solving shape-analysis problems in languages with destructive updating". *ACM Trans. Program. Lang. Syst.* 20(1): 1–50.

Sagiv, S., T. Reps, and R. Wilhelm (1999). "Parametric shape analysis via 3-valued logic". In: *Symposium on Principles of Programming Languages (POPL)*. 105–118.

Sagiv, M., T. Reps, and R. Wilhelm (2002). "Parametric shape analysis via 3-valued logic". *Transactions on Programming Languages and Systems (TOPLAS)*. 24(3): 217–298.

Sharir, M. and A. Pnueli (1981). "Two approaches to interprocedural data flow analysis". In: *Program Flow Analysis: Theory and Applications*. Englewood Cliffs, NJ: Prentice-Hall, Inc. Chap. 7.

Shivers, O. (1991). "Control-flow analysis of higher-order languages". *PhD thesis*. Carnegie Mellon University.

Smaragdakis, Y. and G. Balatsouras (2015). "Pointer analysis". *Found. Trends Program. Lang.* 2(1): 1–69.

Sotin, P. and B. Jeannet (2011). "Precise interprocedural analysis in the presence of pointers to the stack". In: *European Symposium on Programming (ESOP)*. 459–479.

Sotin, P., B. Jeannet, and X. Rival (2010). "Concrete memory models for shape analysis". *Electronic Notes in Theoretical Computer Science*. 267(1): 139–150.

Srivastava, S., N. Immerman, and S. Zilberstein (2011). "A new representation and associated algorithms for generalized planning". *Artificial Intelligence*. 175(2): 615–647.

Steensgaard, B. (1996). "Points-to analysis in almost linear time". In: *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. ACM. 32–41.

Stransky, J. (1992). "A lattice for abstract interpretation of dynamic (LISP-like) structures". *Inf. Comput.* 101(1): 70–102.

Full text available at: http://dx.doi.org/10.1561/2500000037

Toubhans, A., B.-Y. E. Chang, and X. Rival (2013). “Reduced product combination of abstract domains for shapes”. In: *Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*. 375–395.

Toubhans, A., B.-Y. E. Chang, and X. Rival (2014). “An abstract domain combinator for separately conjoining memory abstractions”. In: *Static Analysis Symposium (SAS)*. 285–301.

Vafeiadis, V. (2010). “Automatically proving linearizability”. In: *International Conference on Computer Aided Verification (CAV)*. Ed. by T. Touili, B. Cook, and P. B. Jackson. Vol. 6174. *Lecture Notes in Computer Science*. Springer. 450–464.

Yang, H. (2007). “Towards shape analysis for device drivers”. In: *Verification, Model Checking, and Abstract Interpretation, 8th International Conference, VMCAI 2007, Nice, France, January 14–16, Proceedings*. 267.

Yang, H., O. Lee, J. Berdine, C. Calcagno, B. Cook, D. Distefano, and P. W. O’Hearn (2008). “Scalable shape analysis for systems code”. In: *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7–14, Proceedings*. 385–398.