

University of Technology Sydney

School of Computer Science

Faculty of Engineering and Information Technology

Dissertation submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

under principal supervisor Dr. Yulei Sui and co-supervisor Dr. Shiping Chen

Data Structures for Points-To Analysis

Mohamad Barbar

October 2022

Certificate of Original Authorship

I, Mohamad Barbar, declare that this thesis, is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the School of Computer Science, Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Production Note:

Signature: Signature removed prior to publication.

Date: 06/10/2022

Preface

With the end of my candidature, more than 4 years later, it is time to collect my work in one place. I hope it is of use to some.

Firstly, major thanks is due to my family who were patient with me during my candidature. 4 years is not a short period of time.

I also thank my principal supervisor, Dr. Yulei Sui, for the many long hours of discussion on both the big picture and the smallest of details. Certainly, working with Yulei got me started with static analysis, and it is an area I would like to continue hacking on. The opportunity to work on SVF has also been great as my first prolonged exposure to a large codebase where performance really matters. I also thank my co-supervisor, Dr. Shiping Chen, and CSIRO's Data61 as a whole, for supporting my research with the Data61 scholarship.

My thanks also goes to my peers at UTS who kept me company, particularly before moving to work-from-home at the start of the pandemic; Omar, Mingshan, Ayman, Joakim (who also inspired some of the work in Chapter 4), Ibraheem, Yahya, and Akram. I also thank the Program Analysis Group, especially Yanxin and Yuxiang with whom I started around the same time, though it was unfortunate meetings had to move online. I must also say that all my interactions with administration at UTS has been pleasant, and express my thanks in that direction. At CSIRO, I also thank Ejaz for organising seminars allowing for some nice exchange of ideas. Unfortunately, that too was cut short by the pandemic.

Finally, most of all, I give thanks to the All-Wise, the All-Knowing and hope for His forgiveness for my many shortcoming during this candidature.

MOHAMAD BARBAR
Sydney, Australia
October 2022

Abstract

In this dissertation, we present improvements to data structures, and the algorithms upon them, for points-to analysis. Our focus is mainly on flow-sensitive analysis but our techniques can either be applied to other analyses or used in analyses which combine flow-sensitivity with other sensitivities. For staged flow-sensitive analysis (SFS), we introduce a pre-analysis (meld versioning) where we determine when it is possible to reuse the points-to sets of individual address-taken variables at different program points, then perform the main analysis using this information. Meld versioning is also amenable to parallelisation with minimal effort. For points-to sets, we introduce an improved bit-vector stripping both leading and trailing zero-words, then use that to aid in improving the object-to-identifier mapping required to use bit-vectors as points-to sets. We frame this as an integer programming problem, yielding an optimal solution but with impractical performance, and so we develop a more approximate (yet extremely fast) method based on hierarchical clustering. We also describe hash consing and memoisation, along with some optimisations which would otherwise be impractical, for points-to sets. We have implemented our techniques in open source points-to analysis framework SVF, and upon evaluating with 12 open source programs, we find, on average, a speedup of almost $6\times$ and a reduction in memory usage of more than $3.97\times$ when compared with a baseline SFS.

Publications

Mohamad Barbar, Yulei Sui, and Shiping Chen. 2020. Flow-Sensitive Type-Based Heap Cloning. In 34th European Conference on Object-Oriented Programming (ECOOP '18, Vol. 166). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Germany, 24:1–24:26. <https://doi.org/10.4230/LIPIcs.ECOOP.2020.24>

Mohamad Barbar, Yulei Sui, and Shiping Chen. 2021. Object Versioning for Flow-Sensitive Pointer Analysis. In 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO '21). IEEE Computer Society, USA, 222–235. <https://doi.org/10.1109/CGO51591.2021.9370334>

Mohamad Barbar and Yulei Sui. 2021. Hash Consed Points-To Sets. In International Static Analysis Symposium (SAS '21). Springer, Germany, 25–48. https://doi.org/10.1007/978-3-030-88806-0_2

Mohamad Barbar and Yulei Sui. 2021. Compacting Points-to Sets through Object Clustering. Proceedings of the ACM on Programming Languages 5, OOPSLA, Article 159 (Oct. 2021), 27 pages. <https://doi.org/10.1145/3485547>

Colophon

This dissertation was created using XeLaTeX. The Maggi Memoir Thesis template, originally by Federico Maggi and later modified by Vel from `LaTeXTemplates.com`, is used as a base after some slight modifications. The body is set 10pt with Linux Libertine O primarily. Linux Biolinum O, Latin Modern Math, and Latin Modern Mono make occasional appearances. Graphs, in the graph theoretical sense, are drawn with TikZ/PGF, tables use the `booktabs` package, and the single line graph in this dissertation is drawn with Matplotlib (DejaVu Sans on the axes).

Contents

List of Figures	xvi
List of Tables	xviii
List of Acronyms	xxi
1 Introduction	1
1.1 Contributions	3
2 Background	5
2.1 Program Representation	5
2.2 Inclusion-Based Points-To Analysis	7
2.2.1 Basic Definition	9
2.2.2 Field-Sensitivity	10
2.3 Flow-Sensitive Inclusion-Based Points-To Analysis	12
2.3.1 Basic Definition	12
2.3.2 Staged Flow-Sensitive Analysis	14
2.4 Benchmarks	18
3 Object Versioning	21
3.1 Motivating Example	22
3.2 Meld Labelling	25
3.3 Versioning Objects	28
3.3.1 Preversioning	29
3.3.2 Meld Versioning	32
3.4 Flow-Sensitive Points-To Analysis with Versioned Objects	33
3.5 Efficient Versioning	36
3.5.1 Per-Object Versioning	36

3.5.2	<i>o</i> -SVFG Isomorphism	38
3.5.3	Foregoing Indirect Value-Flow Edges	38
3.5.4	Storing Fewer Versions	39
3.5.5	Parallelisation	40
3.6	Evaluation	40
3.6.1	Time	42
3.6.2	Memory Usage	42
3.6.3	Multithreading	43
3.7	Related Work	44
3.8	Conclusion	45
4	Compacting Points-To Sets	47
4.1	Representing Points-To Sets as Bit-Vectors	49
4.1.1	Contiguous and Sparse Bit-Vectors	50
4.1.2	Core Bit-Vector	53
4.2	Compacting Points-To Sets	57
4.2.1	Integer Programming Formulation	57
4.2.2	Hierarchical Clustering	60
4.2.3	Clustering Objects	63
4.2.4	More Efficient Region-Based Clustering	65
4.2.5	Word-Aligned Identifier Mapping	67
4.3	Evaluation	69
4.3.1	Linkage Criteria and Required Words	70
4.3.2	Time	71
4.3.3	Memory	72
4.4	Related Work	72
4.5	Conclusion	73
5	Hash Consed Points-To Sets	75
5.1	Motivating Example	77
5.2	Approach	79
5.2.1	Hash Consed Points-To Sets	79
5.2.2	Exploiting Set Properties	81
5.3	Evaluation	84
5.3.1	Flow-Insensitive Analysis	85
5.3.2	VSFS	86
5.3.3	Effect of Preemptive Memoisation	88

5.4	Related Work	88
5.5	Conclusion	89
6	Conclusion	91
6.1	Applying the Techniques in this Dissertation Together	91
6.2	Summary	93
6.3	Future Work	94
	Bibliography	95

List of Figures

2.1	Domains our analysed programs operate on.	5
2.2	Instructions we define our analyses upon.	6
2.3	Inference rules for a basic inclusion-based points-to analysis.	9
2.4	Modifications and additions to the inference rules in Figure 2.3 to allow for field-sensitivity.	11
2.5	Inference rules for a flow-sensitive field-sensitive inclusion-based points-to analysis.	13
2.6	Inference rules for the main phase of <i>staged flow-sensitive analysis</i> (SFS).	17
3.1	An example motivating the effectiveness of object versioning.	23
3.2	Meld labelling process. κ_n is the label of node n	26
3.3	An example of meld labelling. Patterns are labels and the meld operator \diamond combines them. The blank pattern is the identity.	27
3.4	The <i>sparse value-flow graph</i> (SVFG) from the motivating example after the preversioning phase. Versions introduced in this phase are boxed.	30
3.5	Preversioning inference rules. $nv(o)$ returns a new version for o and $pt^a(p)$ is the points-to set of p according to the auxiliary analysis.	31
3.6	An example SVFG involving two objects, a and b	32
3.7	Inference rules for meld versioning.	33
3.8	The SVFG from the motivating example after being versioned. Consumed/yielded versions changed during meld versioning are boxed.	33
3.9	Inference rules for the new main phase flow-sensitive analysis in VSFS.	35

3.10	Versioning inference rules using our (qualified) strongly connected components.	37
3.11	Inference rule to determine version reliance.	38
3.12	Inference rule modifications from the main analysis rules in Figure 3.9 to account for the introduction of version reliances and the removal of indirect value-flow edges in the SVFG.	39
4.1	Agglomerative clustering process (b) of coordinate data (a) and the resulting dendrogram (c).	61
4.2	Memory used to represent various (condensed) distance matrices.	65
5.1	Example program fragment in (a), constraints generated for flow-insensitive analysis in (b), operations performed to fulfil the constraints in (c), and final results in (d). Initially, we assume $pt(p) = \{o_1\}$, $pt(q) = \{o_2\}$, $pt(r) = \{o_3, o_4\}$, and remaining points-to sets are empty. Duplicate points-to sets and operations are highlighted in grey	78
5.2	Global pool of points-to sets in (a), the union operations table in (b), and the result in (c) using references instead of concrete points-to sets for the analysis in Figure 5.1.	81

List of Tables

2.1	Variable and SVFG statistics for our benchmark programs.	16
2.2	Information about our benchmark programs.	19
3.1	Time taken (s) and memory usage (GB) of SFS and VSFS.	41
3.2	Time (s) breakdown for VSFS.	42
3.3	Versioning (and total) time (s) for VSFS when versioning is performed with 1, 2, and 4 threads.	43
4.1	Time taken (s) and memory usage (GB) for VSFS using standard bit-vectors and core bit-vectors.	56
4.2	Region statistics for our benchmark programs.	67
4.3	Number of words required (using core bit-vectors) in the theoretical best case, with the original mapping, with the mappings produced by clustering using the single, average, and complete linkage criteria, and the improvement versus the original mapping. The bolded value represents the mapping predicted to be best after the auxiliary analysis, and this is what is compared in the final column.	70
4.4	Time taken (s) and memory usage (GB) for VSFS (using core bit-vectors) without and with a mapping produced by clustering.	71
5.1	Statistics on prevalence of duplicate points-to sets in our benchmark programs.	76
5.2	Time taken (s) and memory usage (GB) for flow-insensitive points-to analysis without and with hash consed points-to sets.	85
5.3	The number of concrete, property, lookup, and preemptive unions for flow-insensitive analysis (and the proportion of the total in parentheses).	86

5.4	Time taken (s) and memory usage (GB) for VSFS without and with hash consed points-to sets.	87
5.5	The number of concrete, property, lookup, and preemptive unions for VSFS (and the proportion of the total in parentheses).	87
6.1	Time taken (s) and memory usage (GB) for SFS (with standard bit-vectors) and all of our techniques combined: VSFS, efficient versioning (4 threads), core bit-vectors, an object-to-identifier mapping produced through clustering, and hash consed points-to sets.	92

List of Acronyms

BDD binary decision diagram	47
DUG def-use graph	15
GLLVM Whole Program LLVM in Go	18
ICFG interprocedural control-flow graph	14
IP integer programming	57
IR intermediate representation	5
ISA instruction set architecture	47
OOM out of memory	20
PAG pointer assignment graph	18
SFS staged flow-sensitive analysis	xvi
SSA static single assignment	6
STL Standard Template Library	18
SVFG sparse value-flow graph	xvi
VSFS versioned staged flow-sensitive analysis	3

