DIPLOMARBEIT

UNIVERSITÄTSZENTRUM INFORMATIK

Martin-Luther-Universität Halle-Wittenberg

Points-To Analysen auf attributtierten Strukturbäumen mit (RO)BDDs

Andreas Both

(2004)

Allgemeine Angaben

Die Diplomarbeit wurde am Lehrstuhl für Software-Engineering und Programmiersprachen von Prof. Dr. Wolf Zimmermann, Institut für Informatik, Fachbereich Mathematik und Informatik der Martin-Luther-Universität angefertigt.

Zur Kontaktaufnahme benutzen Sie bitte die E-Mail-Adresse
direktor@uzi.uni-halle.de.

Zusammenfassung

Durch die immense Verbreitung von Software und der Integration von Software in alle Geschäfts- und Lebensbereiche gilt der Fehlervermeidung bei der Softwareentwicklung besonderes Interesse, da je nach Größe und Lebensdauer bis zu 80 Prozent der Kosten eines Programms bei den Wartungsarbeiten entstehen [1]. Dies gilt, obwohl die Anzahl der Fehler pro tausend Codezeilen in den vergangenen Jahrzehnten stark gesunken ist ([1]), denn der Umfang des verwendeten Quellcodes ist gleichzeitig überpropotional stark gestiegen. Folglich stellt die frühzeitige Fehlererkennung einen wichtigen Ansatz dar, um den wirtschaftlichen Erfolg einer Software nicht zu gefährden bzw. zu steigern.

Eine der größten und am schwersten zu vermeidenden Fehlerquellen – abgesehen von semantischen Fehlern – sind hierbei Laufzeitfehler, ausgelöst durch fehlerhafte Zeiger (z. B. Null-Pointer). Deshalb ist es eine Herausforderung, möglichst viele Informationen über den Quellcode eines Programms und den darin enthaltenen Zeigern zu ermitteln, um sie zur Verhinderung von Fehlern nutzen zu können bzw. potenzielle Fehlerquellen bereits zur Übersetzungszeit (engl. *compile-time*) zu erkennen.

Ein Teil dieser Informationen kann durch *Points-To-Analysen* (im Folgenden *Zeigeranalysen* genannt) ermittelt werden, welche diese danach zur weiteren Verarbeitung an andere Werkzeuge weiterleiten können. Neben der Informationsverarbeitung im Übersetzer (engl. *compiler*) ist es aber ebenso möglich, diese den Programmierern

zur Verfügung zu stellen, um Probleme sichtbar zu machen. Es ist aber auch möglich, die Programmierung durch Integration in integrierte Entwicklungsumgebungen (IDE) zu unterstützen. Ein dabei auftretender positiver Nebeneffekt wäre, dass den Programmierern mehr Informationen über den Kontroll- und Datenfluss sowie Nebeneffekte (engl. *side effects*) dargestellt werden können. Dieses Problem tritt umso mehr in den Vordergrund, da Programme immer weiter wachsen.

Hinzu kommt, dass die Zeiten, in denen ein Programmierer oder eine Programmiererin ganze Programme allein schrieb, vorbei sind. Der heutige Entwickler verfügt nur noch über eine sehr eingeschränkte Sicht auf den Quellcode.

Des Weiteren werden durch den Einsatz von bspw. Komponentensystemen oder Klassenbibliotheken in zunehmendem Maße Teile des Quellcodes verborgen (Blackbox) und Funktionalität auf möglicherweise nicht offensichtliche „callbacks“ ausgelagert. Diese nicht offensichtlichen Informationen gilt es aufzudecken und nutzbar zu machen.

So gut diese Idee ist, so problematisch ist ihre Anwendung in der Praxis. Bisher entwickelte Zeigeranalyse-Algorithmen stoßen bei großen Programmen schnell an die Grenzen bzgl. Laufzeit oder Speicherbedarf (vgl. [2]) und werden damit für den Anwender unbrauchbar. Diesen Problemen wurde versucht durch verschiedene Adaptionen entgegenzuwirken, welche aber zumeist Einbußen an der Präzision der Ergebnisse nach sich zogen (vgl. [3], [4]). Der Speicherplatzbedarf konnte aber dadurch (teilweise nur in der Praxis) bis zu einem linearen Vielfachen der Größe des zu analysierenden Programms beschränkt werden.

Eine Idee, dem Speicherplatzproblem durch den Einsatz von Binären Entscheidungsgraphen (engl. *Binary Decision Diagrams, BDD*) entgegen zu wirken, haben Berndl, Lhoták, Qian, Hendren und Umanee ([5]) mittels einer Adaption des Algorithmus von Andersen ([6]) für Java vorgestellt. Diese Idee bildet die Grundlage dieser Arbeit.

In dieser Diplomarbeit wurde untersucht, ob es möglich ist, bekannte Zeigeranalyse-Algorithmen so umzugestalten, dass es möglich ist (fast) nur BDD-Operationen zur Berechnung der Zeiger-Informationen zu verwenden. Im Einzelnen wurden folgende bekannte Algorithmen unter Benutzung von BDDs implementiert:

- der flussin- und kontextinsensitive Algorithmus von Andersen ([6]),
- der flussin- und kontextinsensitive Algorithmus von Shapiro und Horwitz ([7]),
- der flussin- und kontextsensitive Algorithmus von Liang und Harrold ([2]).

Darüber hinaus wurde ein neuer fluss- und kontextsensitiver Algorithmus entworfen und vorgestellt.

Als Basis für die Implementierung diente das (RO)BDD-Paket BuDDy ([7]). Der attributierte Strukturbaum jedes zu analysierenden Programms wird durch das Übersetzerbauwerkzeug eli ([8]) erzeugt. Dadurch wird praktisch eine Sprachunabhängigkeit gewährleistet, die es ermöglicht, jederzeit Programme verschiedener Programmiersprachen zu analysieren.

Diese Zeiger-Algorithmen wurden in einer Bibliothek unter dem Namen PoTABu (Points-To Analyses using BuDDy) zusammengefasst, um eine Weiterverwendung in anderen Arbeiten zu ermöglichen. Darüber hinaus wurde eine grafische Benutzeroberfläche (PoTABuGUI) entwickelt, welche die Auswahl eines Algorithmus und der Parameter intuitiv ermöglicht.

Wie in dieser Arbeit gezeigt wurde, stellt die Verwendung von BDDs eine gute Möglichkeit zur Lösung der Speicherprobleme dar und stellt zusätzlich noch effiziente Operationen zur Verfügung, die die Entwicklung neuer Algorithmen vereinfachen. Durch den Einsatz von BDDs ist ein höherer Ansatz bei der Spezifikation und Entwicklung von Zeigeranalyse-Algorithmen möglich.

Literaturverzeichnis

[1] Balzert, Helmut, *Lehrbuch der Software-Technik – Software-Entwicklung*, Spektrum Akademischer Verlag, 2001

[1] Balzert, Helmut, *Lehrbuch der Software-Technik – Software-Entwicklung*, Spektrum Akademischer Verlag, 2001

[2] Donglin Liang and Mary Jean Harrold, *Efficient points-to analysis for whole-program analysis*, Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering, 1999

[3] Bjarne Steensgaard, *Points-to analysis in almost linear time*, Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 1996

[4] Marc Shapiro and Susan Horwitz, *Fast and accurate flow-insensitive points-to analysis*, Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 1997

[5] Marc Berndl and Ondrej Lhoták and Feng Qian and Laurie Hendren and Navindra Umanee, *Points-to analysis using BDDs*, Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation, 2003

[6] Lars Ole Andersen, *Program Analysis and Specialization for the C Programming Language*, DIKU, University of Copenhagen, 1994

[7] Jørn Lind-Nielsen, *BuDDy - A Binary Decision Diagram Package*, http://buddy.sourceforge.net, 1996

[8] Universität von Colorado in Boulder (USA), Universität-GH Paderborn (Deutschland), Macquarie Universität in Sydney (Australien), *eli,* http://eli-project.sourceforge.net