TECHNISCHE
UNIVERSITÄT
DARMSTADT

Synthesizing Program Generators for Embedded Software

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

Dissertation

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)

von

Michael Jung, M.S./Univ. of Colorado

geboren in Weilburg

Referenten der Arbeit:	Prof. Dr.-Ing. S. A. Huss
	Prof. Dr. rer. nat. A. Schürr
Tag der Einreichung:	21.05.2007
Tag der mündlichen Prüfung:	18.07.2007

Darmstadt 2007
D17

Berichte aus der Informatik

Michael Jung

Synthesizing Program Generators for Embedded Software

D 17 (Diss. TU Darmstadt)

Shaker Verlag
Aachen 2007

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Zugl.: Darmstadt, Techn. Univ., Diss., 2007

Printed in Germany.

ISBN 978-3-8322-6586-1
ISSN 0945-0807

Shaker Verlag GmbH • P.O. BOX 101818 • D-52018 Aachen
Phone: 0049/2407/9596-0 • Telefax: 0049/2407/9596-9
Internet: www.shaker.de • e-mail: info@shaker.de

In Gedenken an

Maria und Heinrich Jung

Zusammenfassung

Eingebettete Systeme werden oft in großen Stückzahlen produziert und sollen möglichst geringe Stückkosten aufweisen. Aus diesem Grund muss die zugehörige Software sehr effizient verfügbare Hardwareressourcen nutzen. Dies verhindert den Einsatz von generischen Softwarekomponenten, die zusätzliche Ressourcen beanspruchen, um flexibel und anpassbar zu sein.

Generatoren, die aus einer vorgegebenen Konfiguration angepassten Quelltext erzeugen, kombinieren die Flexibilität generischer mit der Effizienz spezialisierter Software. Die Entwicklung eines Generators ist jedoch ungleich aufwendiger als die eines generischen Softwaremoduls. Die vorliegende Dissertation präsentiert *PEAC* – ein Werkzeug, das die anspruchsvolle Generatorenentwicklung vereinfacht – und illustriert die zu Grunde liegenden Algorithmen.

PEAC basiert auf dem Konzept der partiellen Evaluation: Unter Angabe einer Menge von Eingangsparametern des generischen Programms wird durch vorgezogene Auswertung abhängiger Ausdrücke und Anweisungen eine spezialisierte Variante bestimmt, die im Allgemeinen Rechenleistung und Speicher geringer beansprucht.

Der Prozess der partiellen Evaluation wird mittels einer Folge von Analysealgorithmen implementiert. Hierbei wird zunächst anhand des abstrakten Syntaxbaums des generischen Programms eine Points-to Analyse durchgeführt. Dies ist notwendig, um auch für Programme, die Zeigervariablen verwenden, gute Spezialisierungsergebnisse zu erzielen. Die im Rahmen dieser Promotion entwickelte und in *PEAC* implementierte Points-to Analyse bestimmt schärfere Abschätzungen als bekannte Analysen ähnlicher Komplexität.

Der Entwickler gibt anschließend eine Aufteilung der Programmparameter in zwei Teilmengen vor, von denen eine die Parameter enthält, die während des Betriebs des eingebetteten Systems variabel sind. Die verbleibende Teilmenge enthält die statischen Parameter, deren Werte bereits zur Laufzeit des Generators bekannt sind. Basierend auf dieser Aufteilung markiert die im zweiten Schritt auszuführende Binding-Time Analyse im abstrakten Syntaxbaum alle Ausdrücke und Anweisungen, die der Generator vorzeitig auswerten kann. Diese für *PEAC* entwickelte Analyse verwendet abstrakte Interpretation, deckt nahezu den kompletten Umfang der Sprache C ab und minimiert durch die Berechung einer monovarianten Annotation die potentielle Gefahr, Quelltexte ausufernder Größe zu erzeugen.

Anhand der Markierungen des abstrakten Syntaxbaums wird in einem anschließenden Schritt ein Generator erzeugt. Während dessen Ausführung werden die statischen

Ausdrücke und Anweisungen ausgewertet. Dies muss mit der Semantik der Zielplattform geschehen, die im Allgemeinen nicht der Entwicklungsplattform entspricht. Diese Anforderung, die nur bei der Entwicklung eingebetteter Systeme besteht, wird dadurch erfüllt, dass *PEAC* eine konfigurierbare Emulationsbibliothek nutzt. Dies ist ein weiteres Alleinstellungsmerkmal von *PEAC*.

Die Dissertation wird mit einer Fallstudie abgeschlossen, durch die die Tauglichkeit des vorgestellten Ansatzes nachgewiesen wird.

Abstract

Embedded systems are often produced in high volume and are to exhibit low unit costs. For that reason it is important that system software utilizes the available hardware resources very efficiently. This prohibits the adoption of generic reusable software components that demand additional resources to achieve flexibility and adaptability.

By computing specialized source code based on a given configuration, Generators combine the flexibility of generic software with the efficiency of specialized code. However, it requires a far higher effort to implement a generator than a generic software module. This thesis presents *PEAC* – a tool, which eases the challenge of generator development – and illustrates its base algorithms.

PEAC is based on the concept of Partial Evaluation: Given a set of input parameters of the generic program, a specialized version is derived by early evaluation of dependent expressions and statements. Generally, this specialized code is less demanding in terms of processing time and memory.

The Partial Evaluation process is implemented as a sequence of analyzes. Initially, a Points-to Analysis is applied to the generic program's abstract syntax tree. This is necessary to achieve good specialization results even for programs that apply pointer variables. The analysis developed for and implemented in *PEAC* gives better approximations than similar analyzes of comparable computational complexity.

Afterwards the developer provides a partitioning of the program's parameters into two subsets. One set contains the parameters that will be dynamic at system run-time. The other one holds the static parameters, whose values will already be known at system design time when the generator is applied. Based on this partitioning, the Binding-Time Analysis annotates all statements and expressions in the abstract syntax tree, which turn out to be pre-evaluable. *PEAC*'s analysis applies Abstract Interpretation, handles virtually the complete C language and minimizes the probability of code bloat problems by computing a mono-variant annotation.

Given these abstract syntax tree annotations, the generator is derived in a subsequent step. During its execution the static expressions and statments are evaluated. This has to be done with the semantics of the target platform, which typically differs from the host platform. This requirement, which is unique to embedded systems development, is met by the application of a widely configurable emulation library. This is another feature, which sets *PEAC* apart from other Partial Evaluators.

The thesis concludes with a case study, which demonstrates the effectiveness of the proposed approach.

Acknowledgments

During my employment as research assistant with the Integrated Circuits and Systems Laboratory at Technische Universität Darmstadt, I have had the pleasure to work with a lot of wonderful people, which was essential in making this a productive and enjoyable time.

I am especially grateful to Prof. Dr. Sorin A. Huss, for the excellent supervision, for giving encouragement and motivation, for many prolific discussions and for his open mindedness towards my ideas.

A very special thanks goes to Prof. Dr. Andy Schürr for refereeing this thesis and for his interest in my work.

I would also like to thank GM Powertrain Europe for funding the project, which led to this thesis. This project was initiated by Matthias Deegener and Gerhard Landsmann. Many thanks for their interest and for giving enough freedom to develop the project ideas further.

A grateful word of thanks to all my colleagues at the Integrated Circuits and Systems Laboratory, for their support, cooperation, and enjoyable coffee break discussions. Maxim Anikeev, Tom Assmuth, Prih Hastono, Stephan Hermanns, Dan Honciuc, Elisabeth Hudson, Stephan Klaus, Andreas Kühn, Ralf Laue, Felix Madlener, Gregor H. Molter, Tim Sander, Abdul Shoufan, Maria Tiedemann, Jürgen Weber, Song Yuan, and Kaiping Zeng. And special thanks to Markus Ernst and Steffen Klupsch for being great mentors.

Many thanks to the students I have had the joy to work with, for their dedication and constructive discussions. Jens Bieger, Daniel Kutzmann, Ralf Laue, Christoph Reeg, Sven Simon, Andreas Sommerfeld, and Thomas Steiniger.

I would also like to thank my parents Beate and Theo Jung for their love and encouragement, and for the support they provided me through my entire life.

Last, but not least, very special thanks to my love and best friend Yvonne Weber. For continuous and profound support, for relinquishing time with me when I had to work on weekends, and for the wonderful years with her.

Contents