GOETHE UNIVERSITÄT FRANKFURT AM MAIN

www.uni-frankfurt.de

Seminar:

Spezielle Themen zu Softwaresystemen

Vorbesprechung

PD Dr. David Sabel

Sommersemester 2018

Büro, Email, Webseite...

- Büro: Zimmer 118, Robert-Mayer-Str. 11-15
- Email: `sabel@ki.informatik.uni-frankfurt.de`
- Webseite zum Seminar: `goethe.link/SIW-I-S-2018`

Termine: nach Vereinbarung, im Doodle

	Mo	Di	Mi	Do	Fr

	18.6.	19.6.	20.6.	21.6.	22.6.
08-12					
14-18					

	25.6.	26.6.	27.6.	28.6.	29.6.
08-12					
14-18					

	3.7.	4.7.
08-12		
14-18		

Abgabe der Ausarbeitung: 2 Wochen vor den Vorträgen

Modul im Masterstudiengang

- Veranstaltungsnummer SIW-I-S, verwendbar in den Modulen M-SIW-HSA-S (SIW-Seminar A) und M-SIW-HSB-S (SIW-Seminar B)
- Spezialisierung „Softwaresysteme,Informationsverarbeitung und Wissensverarbeitung“
- Umfang des Moduls: 5 CP
- Anmeldung durch den Veranstalter beim Prüfungsamt bis 20. April

Prüfung

- Regelmäßige Teilnahme
- schriftliche Ausarbeitung (Note 4,0 oder besser)
- sowie ein Vortrag (Note 4,0 oder besser)
- Note: Arithmetisches Mittel aus den beiden Noten

Vortrag und Ausarbeitung

Vortrag

- ca 60min Dauer
- **keine** Kopie der Ausarbeitung!
- Beamer + Notebook

Ausarbeitung

- Abgabe **zwei Wochen vor** dem Vortrag
- Ausdruck **und** PDF
- Umfang ca.**15 Seiten**

Übersicht

1. Algebraic translations, correctness and algebraic compiler construction
2. Observational program calculi and the correctness of translations
3. On sessions and infinite data
4. Biorthogonality, step-indexing and compiler correctness
5. Proving correctness of a compiler using step-indexed logical relations
6. Lightweight verification of separate compilation
7. Fully-abstract compilation by approximate back-translation
8. Refinement reflection: complete verification with SMT
9. Online detection of effectively callback free objects with applications to smart contracts
10. Linear Haskell: practical linearity in a higher-order polymorphic language

- Algebraic translation methods, examples from the fields compiler construction, database updates, concurrent programming languages, logic, natural language translation, and natural language semantics.
- notion of 'correctness of translation'
- discussion on which correctness notion is the 'right' one

Literatur:

Theo M.V. Janssen: Algebraic translations, correctness and algebraic compiler construction. In Theor. Comput. Sci. , Volume 199, S. 25-56, 1998

- translations between programming languages
- languages with observational (or contextual) semantics
- what are the relevant questions concerning correctness of translations
- exemplary translations demonstrate the notions

Literatur:

Manfred Schmidt-Schauß, David Sabel, Joachim Niehren, and Jan Schwinghammer: Observational program calculi and the correctness of translations. In Theor. Comput. Sci. , Volume 577, S. 98-124 2015

Thema 3: On sessions and infinite data

- programm distributed computations
- over infinite data structures
- calculus with call-by-name functioncal core and session-based communication
- properties: normalisation, progress of processes, etc.

Literatur:

Paula Severi, Luca Padovani, Emilio Tuosto and Mariangiola Dezani-Ciancaglini, On Sessions and Infinite Data, In Coordination Models and Languages - 18th IFIP WG 6.1 International Conference, COORDINATION 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings, S. 245–261, 2016

Thema 4: Biorthogonality, step-indexing and compiler correctness

- logical relations between the donotational semantics of two languages: a simply typed functional language with recursion low-level language of the SECD machine
- The relations capture can be used to prove correctness of a (simple) compiler
- The results have been formalized in the Coq proof assistant.

Literatur:

Nick Benton and Chung-Kil Hur, Biorthogonality, Step-indexing and Compiler Correctness In Proceedings of the 14th ACM SIGPLAN International Conference on Functional Programming, S. 97-108,

- correctness of a compiler for a call-by-name language using logical relations
- simply typed lambda calculus with recursion is translated into Krivine machine code
- We formalized the proof in the Coq proof assistant.

Literatur:

Proving Correctness of a Compiler Using Step-indexed Logical Relations Leonardo Rodríguez , Miguel Pagano , Daniel Fridlender In Proceedings of the Tenth Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2015), Electronic Notes in Theoretical Computer Science Volume 323, 11 July 2016, S. 197 – 214

Thema 6: Lightweight verification of separate compilation

- how to verify the correctness of separate compilation
- compositional notions of compiler correctness
- develop several lightweight techniques that recast the compositional verification problem in terms of whole-program compilation
- The result is SepCompCert, the first verification of separate compilation for the full CompCert compiler.

Literatur:

Jeehoon Kang, Yoonseung Kim, Chung-Kil Hur, Derek Dreyer, Viktor Vafeiadis, Viktor: Lightweight Verification of Separate Compilation In Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, S. 178 – 190, 2016

Thema 7: Fully-abstract compilation by approximate back-translation

- Fully-abstract: compilation from source language programs to target language programs reflects and preserves behavioural equivalence.
- A common proof technique is based on the back-translation of target-level program contexts to source-level contexts.
- The key insight is that it suffices to construct an approximate back-translation.

Literatur:

Dominique Devriese, Marco Patrignani, and Frank Piessens: Fully-abstract Compilation by Approximate Back-translation, Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '16), S. 164-177, 2016.

Thema 8: Refinement reflection: complete verification with SMT

- Refinement Reflection: a new framework for building SMT-based deductive verifiers.
- The key idea is to reflect the code implementing a user-defined function into the function's (output) refinement type.
- we show that reflection permits the specification of arbitrary functional correctness properties.
- We have implemented reflection in Liquid Haskell

Literatur:

Niki Vazou, Anish Tondwalkar, Vikraman Choudhury, Ryan G. Scott, Ryan R. Newton, Philip Wadler, Ranjit Jhala.2018. Refinement Reflection: Complete Verification with SMT, Proc. ACM Program. Lang., Vol. POPL-2, 53:1–53:31.

- Callbacks are essential in many programming environments, but drastically complicate program understanding
- famous DAO bug in the cryptocurrency framework Ethereum employed callbacks to steal $150M.
- Effectively Callback Free (ECF) objects to allow callbacks without preventing modular reasoning
- decidability of dynamically checking ECF
- dynamically checking ECF in Ethereum is feasible and can be done online

Shelly Grossman, Ittai Abraham, Guy Golan-Gueta, Yan Michalevsky, Noam Rinetzky, Mooly Sagiv, Yoni Zohar. Online detection of effectively callback free objects with applications to smart contracts, Proc. ACM Program. Lang., Vol. POPL-2, 48:1–48:28.

- Linear type systems: a function is "linear" if it consumes its argument exactly once. A linear type system gives a static guarantee that a claimed linear function really is linear.
- integrate with existing languages such as OCaml or Haskell.
- demonstrate two kinds of applications of linear types: mutable data with pure interfaces; and enforcing protocols in I/O-performing functions.

Literatur:

Jean-Philippe Bernardy, Mathieu Boespflug, Ryan R. Newton, Simon Peyton Jones, Arnaud Spiwack. Linear Haskell: practical linearity in a higher-order polymorphic language. Proc. ACM Program. Lang., Vol. POPL-2, 5:1–5:29.