Curriculum vitae

Torben Amtoft
email: tamtoft HAT k-state DOT edu
URL: people.cs.ksu.edu/~tamtoft

June 19, 2018

Contents

Academic Record

1993 June 21th: Awarded the Ph.D. degree in Computer Science at DAIMI, University of Aarhus. Advisor: Brian H. Mayoh.

1989 August 18th: Graduated as "cand.scient" (corresponding to an M.Sc.) in Computer Science at DIKU, Copenhagen. Advisor: Neil D. Jones.

1985 Completed "bifag" (corresponding to a Bachelor's degree) in Computer Science and in Mathematics, at the University of Copenhagen.

Current Research Interests

Program analysis, in particular dependency analysis, with applications to language-based security and program slicing.

Awards and Honors

October 2008 was co-awarded a grant of $3M over 5 years from AFOSR (Air Force Office of Scientific Research) for the proposal *Evidence-based Trust in Large-scale MLS Systems.* (John Hatcliff was the PI, other co-PI's were Andrew Appel and Edward Felton from Princeton, Anindya Banerjee, Xinming Ou, Robby.)

January 2006 was co-awarded a grant of $450K over 3 years from AFOSR (Air Force Office of Scientific Research) for the proposal *An Integrated Specification and Verification Environment for Component-based Architectures of Large-scale Distributed Systems.* (John Hatcliff was the PI, Anindya Banerjee was also co-PI.)

August 2004 received the Best Paper Award for the 2004 edition of Static Analysis Symposium (SAS) together with Anindya Banerjee, for our paper *Information Flow Analysis in Logical Form.*

Professional Employments

2008-present Associate professor (tenured) at Kansas State University.

2002-2008 Assistant professor at Kansas State University.

2002 Research associate at Heriot-Watt University,.

1999-2002 Research associate at Boston University, working on the NSF-funded Church Projecct.

1992-1998 Research assistant/research associate at DAIMI, University of Aarhus, working with Hanne Riis Nielson and Flemming Nielson on the LOMAPS project, and latest with Olivier Danvy on BRICS.

1989-1992 Ph.D. scholarship from University of Aarhus, including duties as teaching assistant.

1985-1989 Part time teaching assistant at the University of Copenhagen.

Teaching

Undergraduate Courses

Logical Foundations of Programming, Kansas State University, CIS 301, each Fall during the years 2011-2014, and also most semesters during the years 2002-2008.

Introduction to Programming Languages, Kansas State University, CIS 505, each Fall since 2015; also Fall 2010 (with Xinming Ou) and Fall 2009 (with David Schmidt).

Introduction to Algorithm Analysis, Kansas State University, CIS 575, each Spring since 2013.

Compiler Design Theory, Boston University, CS 525, Spring 2001 (with Assaf Kfoury).

Graduate Courses & Seminars

Programming Languages, Kansas State University, CIS 705, each Fall since 2015 (joint with the undergraduate version, CIS505).

Database Management Systems, Kansas State University, CIS 761, each Spring from 2004 to 2012.

Formal Language Theory, Kansas State University, CIS 770, each Spring from 2009 to 2015.

Software Specification, Kansas State University, CIS 771, each Spring from 2016 to 2018.

Analysis of Algorithms, Kansas State University, CIS 775, each Fall since 2008 (except 2017).

Semantics of Programming Languages, Kansas State University, Spring 2017.

Program Analysis, Kansas State University, CIS 905, Spring 2005 (with Anindya Banerjee) and Spring 2003; CIS 890, Fall 2010.

Language Based Security, Kansas State University, CIS 890, Fall 2004 and Fall 2003 (both with Anindya Banerjee).

Programming the Web/Internet, Boston University, Fall 1999 & Spring 2000 & Fall 2000, with Assaf Kfoury and Santiago Pericas.

Functional Languages, University of Aarhus, Spring 1993 (with Flemming Nielson).

Students

Ph.D. students

Joydeep Mitra (since August 2014, with Venkatesh Ranganath becoming major professor in December 2016).

Committees for Ph.D. students

Zhi Zhang (June 2016, major professor: John Hatcliff), Nic Herndon (April 2016, Doina Caragea), Ming Yang (December 2015, Bill Hsu), Hao Qian (September 2015, Dan Andresen), Ana Stanescu (June 2015, Doina Caragea), Rohit Parimi (June 2015, Doina Caragea), Karthik Tangirala (April 2015, Doina Caragea), Huang Zhu (July 2014, Gurdip Singh), Sumeet Gujrati (November 2013, Gurdip Singh), Waleed Aljandal (December 2008, Bill Hsu), Oksana Tkachuk (December 2008, Matt Dwyer & John Hatcliff), William Deng (June 2007, John Hatcliff & Robby).

Pending: HongMin Li (Doina Caragea), Deepti Lamba (Doina Caragea), Qais Tasali (Eugene Vasserman), Chendi Cao (Mitch Neilsen).

Master students

Master theses: Joshua Donnoe (April 2018).

Master reports: Kaushik Atchuta (April 2014), Balaji Rayakota (April 2013), Dayou Jiang (April 2013).

Master of Software Engineering (MSE) projects: Srunokshi K.P. Neelakantan (July 2010–April 2011), Abhilash Manne (April–December 2010), Phaninder Surapaneni (April–December 2009), Vamsi Mummaneni (August–December 2008), Santosh Bejjamshety (December 2007–August 2008), Nayan Ancha (June–August 2008), Sandhya Bathini (August 2008).

Project Supervision

Independent studies: Joydeep Mitra (2015 & 2016), Shailaja Maddala (2014/15), Balaji Rayakota (2013), Vineet Tadakamalla (2010), Gaurav Chauhan (2008/9), Sandhya Bathini (2008), Santosh Bejjamshety (2008), Anupam Godbole (2008), Venkata Sri Vatsav Reddy Konreddy (2008), Srunokshi K.P. Neelakantan (2008).

Implementation projects: Joshua Donnoe (2016), Deepti Garlapati (2014), Akash Suryawanshi (2014), Nikhita Addanki (2013), Balaji Rayakota (2011 & 2013), Vineet Tadakamalla (2010 & 2011), Naga Sowjanya Karumuri (2008), Abhilash Manne (2008).

Internship reports: Nisha Stephen (2008 & 2009), Nayan Ancha (2008), Greeshma Malgireddy (2008), Aditi Breed (2007).

Senior projects: Chloe Henderson (2018), Anthony Atkinson (2018), Daniel Jones (2016), Jake Ehrlich (2016), Aaron Schif (2016), Joshua Donnoe (2016), Chad Bachman (2014).

Research Advising/Collaboration

Max Wiens-Evangelista (2018-), Matt Link (2017-18), Joydeep Mitra (2014-), Zhi Zhang (2010-2016, [12]) Andrew Cousino (2009-2013), Joey Dodds (2010-2012, [12]), Balaji Rayakota (2010-2011), Vineet Tadakamalla (2010-2011), Edwin Rodriguez (2005-2010, [15, 13]), Ye Zhang (2008-2010, [14]), Jonathan Hoag (2007-2009), Scott Harmon (2006), Sruthi Bandhakavi (2004-2005, [17]), Venkatesh Ranganath (2004-2005, [18, 5]), Oksana Tkachuk (2003).

Publications

[**Monographs**]

[1] Torben Amtoft, Flemming Nielson, and Hanne Riis Nielson. *Type and Effect Systems: Behaviours for Concurrency*. Imperial College Press, 1999.

[**Journal papers**]

[2] Torben Amtoft, Kelly Androutsopoulos, David Clark, Mark Harman, and Zheng Li. An alternative characterization of weak order dependence. *Information Processing Letters*, 110:939–943, October 2010.

[3] Torben Amtoft. Flow-sensitive type systems and the ambient calculus. *Higher-Order and Symbolic Computation*, 21(4):411–442, December 2008.

[4] Torben Amtoft. Slicing for modern program structures: a theory for eliminating irrelevant loops. *Information Processing Letters*, 106(2):45–51, April 2008.

[5] Venkatesh Prasad Ranganath, Torben Amtoft, Anindya Banerjee, John Hatcliff, and Matthew B. Dwyer. A new foundation for control dependence and slicing for modern program structures. *ACM TOPLAS*, 29(5), 2007. A special issue with extended versions of selected papers from the 14th European Symposium on Programming (ESOP 2005).

[6] Torben Amtoft and Anindya Banerjee. A logic for information flow analysis with an application to forward slicing of simple imperative programs. *Science of Computer Programming*, 64(1):3–28, 2007.

[7] Torben Amtoft, Assaf J. Kfoury, and Santiago M. Pericas-Geertsen. Orderly communication in the ambient calculus. *Computer Languages, Systems & Structures*, 28:29–60, 2002 (Elsevier Science).

[8] Torben Amtoft, Hanne Riis Nielson, and Flemming Nielson. Behaviour analysis for validating communication patterns. *Software Tools for Technology Transfer*, 2(1):13–28, 1998.

[9] Torben Amtoft, Flemming Nielson, and Hanne Riis Nielson. Type and behaviour reconstruction for higher-order concurrent programs. *Journal of Functional Programming*, 7(3):321–347, May 1997.

[10] Torben Amtoft and Jesper Larsson Träff. Partial memoization for obtaining linear time behavior of a 2DPDA. *Theoretical Computer Science*, 98(2):347–356, May 1992.

[**Conference papers**]

[11] Torben Amtoft and Anindya Banerjee. A theory of slicing for probabilistic control flow graphs. In *Proc. FoSSaCS 2016* (part of ETAPS'16), pages 180–196, Springer LNCS 9634, 2016. Acceptance rate: 36.5 % A preliminary version, with full proofs, appears as Technical Report 2015-1, CIS Department, Kansas State University, July 2015.

[12] Torben Amtoft, Josiah Dodds, Zhi Zhang, Andrew Appel, Lennart Beringer, John Hatcliff, Xinming Ou, and Andrew Cousino. A certificate infrastructure for machine-checked proofs of conditional information flow. In First Conference on Principles of Security and Trust (part of ETAPS 2012), pages 369–389, Springer LNCS 7215, 2012. Acceptance rate: 30 %.

[13] Torben Amtoft, John Hatcliff and Edwin Rodríguez. Precise and automated contract-based reasoning for verification and certification of information flow properties of programs with arrays. In *Proc. ESOP 2010* (part of *ETAPS 2010*), pages 43–63, Springer LNCS 6012, 2010. Acceptance rate: 24.8 %.

[14] Ye Zhang, Torben Amtoft, and Flemming Nielson. From generic to specific: off-line optimization for a general constraint solver. In Proceedings of 7th International Conference on Generative Programming and Component Engineering (GPCE'08), pages 45–53, ACM Press, October 2008. Acceptance rate: 31 %.

[15] Torben Amtoft, John Hatcliff, Edwin Rodríguez, Robby, Jonathan Hoag, and David Greve. Specification and checking of software contracts for conditional information flow. In Proceedings of the 15th International Symposium on Formal Methods (FM'08), pages 229–245, Springer LNCS 5014, May 2008. Acceptance rate: 21.7 %. An extended version appears as Technical Report SAnToS-TR2007-5, CIS Department, Kansas State University. An extended version appears as Chapter 12 in *Design and Verification of Microprocessor Systems for High-Assurance Applications*, edited by David S. Hardin.

[16] Torben Amtoft and Anindya Banerjee. Verification condition generation for conditional information flow. In Proceedings of the 5th ACM

Workshop on Formal Methods in Security Engineering (FMSE'07), pages 2–11, George Mason University, November 2007. Acceptance rate: 28.6 %. An extended version appears as Technical Report 2007-2, Dept. of Computing and Information Sciences, Kansas State University, August 2007.

[17] Torben Amtoft, Sruthi Bandhakavi, and Anindya Banerjee. A logic for information flow in object-oriented programs. In Proceedings of the 33rd Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL'06), pages 91–102, ACM Press, 2006. Acceptance rate: 19.8 %.

[18] Venkatesh Ranganath, Torben Amtoft, Anindya Banerjee, Matthew B. Dwyer, and John Hatcliff. A new foundation for control-dependence and slicing for modern program structures. In *Proc. ESOP 2005* (part of *ETAPS 2005*), pages 77–93, Springer LNCS 3444, 2005. Acceptance rate: 24.6 %.

[19] Torben Amtoft and Anindya Banerjee. Information flow analysis in logical form. In *Proc. SAS 2004*, pages 100–115, Springer LNCS 3148, 2004. Received the SAS'04 Best Paper Award. Acceptance rate: 36.5 %. An extended version appears as the technical report 2004-3, Department of Computing and Information Sciences, Kansas State University, April 2004.

[20] Torben Amtoft and Henning Makholm and J. B. Wells. PolyA: true type polymorphism for mobile ambients. In *Proc. TCS 2004*, pages 591–604, Kluwer, 2004. An extended version appears as the technical report HW-MACS-TR-0015, School of Mathematical and Computer Sciences, Heriot-Watt University, February 2004.

[21] Torben Amtoft and Robert Muller. Inferring annotated types for interprocedural register allocation with constructor flattening. Proceedings of ACM SIGPLAN TLDI'03 Workshop, pages 86–97, ACM Press, January 2003. Acceptance rate: 42 %.

[22] Torben Amtoft, Assaf J. Kfoury, and Santiago M. Pericas-Geertsen. What are polymorphically-typed ambients? In *Proc. ESOP 2001* (part of *ETAPS 2001*), pages 206–220, Springer LNCS 2028, 2001. Acceptance rate: 34 %. An extended version appears as the technical report BUCS-TR-2000-021, Boston University.

[23] Torben Amtoft and Franklyn Turbak. Faithful translations between polyvariant flows and polymorphic types. In *Proc. ESOP 2000* (part of *ETAPS 2000*), pages 26–40, Springer LNCS 1782, 2000. Acceptance rate: 31 %.

[24] Hanne Riis Nielson, Torben Amtoft, and Flemming Nielson. Behaviour analysis and safety conditions: a case study in CML. In *Proc. FASE'98* (part of *ETAPS'98*), pages 255–269, Springer LNCS 1382, 1998. Acceptance rate: 31 %.

[25] Torben Amtoft. Local type reconstruction by means of symbolic fixed point iteration. In *Proc. ESOP'94*, pages 43–57, Springer LNCS 788, 1994. Acceptance rate: 28 %.

[26] Torben Amtoft. Minimal thunkification. In *Proc. WSA'93*, pages 218–229, Springer LNCS 724, 1993. Acceptance rate: 29 %.

[27] Torben Amtoft. Unfold/fold transformations preserving termination properties. In *Proc. PLILP'92*, pages 187–201, Springer LNCS 631, August 1992. Acceptance rate: 35 %.

[28] Torben Amtoft. Properties of unfolding-based meta-level systems. In *Partial Evaluation and Semantics-Based Program Manipulation (PEPM'91), New Haven, Connecticut*. Sigplan Notices, vol. 26, no. 9, pages 243–254, 1991. Acceptance rate: 41 %.

[29] Torben Amtoft, Thomas Nikolajsen, Jesper Larsson Träff, and Neil D. Jones. Experiments with implementations of two theoretical constructions. In *Logic at Botik, USSR*, pages 119–133, Springer LNCS 363, July 1989.

[Other reviewed papers]

[30] Torben Amtoft, Charles Consel, Olivier Danvy, and Karoline Malmkjær. The abstraction and instantiation of string-matching programs. In *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, Torben Mogensen and David Schmidt and I. Hal Sudborough (editors), pages 332–357, Springer LNCS 2566, 2002. An extended version appeared as Technical Report BRICS RS-01-12, DAIMI, Aarhus, Denmark, April 2001.

[31] Hanne Riis Nielson, Flemming Nielson, and Torben Amtoft. Polymorphic subtyping for effect analysis: the static semantics. In *Analysis*

and Verification of Multiple-Agent Languages, pages 141–171, Springer LNCS 1192, 1997. Acceptance rate: 87 %.

[32] Torben Amtoft, Flemming Nielson, Hanne Riis Nielson, and Jürgen Ammann. Polymorphic subtyping for effect analysis: the dynamic semantics. In *Analysis and Verification of Multiple-Agent Languages*, pages 172–206, Springer LNCS 1192, 1997. Acceptance rate: 87 %.

[33] Flemming Nielson, Hanne Riis Nielson, and Torben Amtoft. Polymorphic subtyping for effect analysis: the algorithm. In *Analysis and Verification of Multiple-Agent Languages*, pages 207–243, Springer LNCS 1192, 1997. Acceptance rate: 87 %.

[**Miscellaneous**]

[34] Torben Amtoft and Anindya Banerjee. A semantics for probabilistic control flow graphs. Submitted for publication, November 2017.

[35] Torben Amtoft and Anindya Banerjee. A theory of slicing for probabilistic control-flow graphs. Submitted for publication, November 2017. An extended version of [11].

[36] Torben Amtoft and Kelly Androutsopoulos and David Clark. Correctly slicing extended finite state machines. Submitted for publication, June 2018. A much preliminary version appears as Research Note RN/13/22 from University College London, Department of Computer Science, December 2013.

[37] Torben Amtoft and J.B. Wells. Mobile processes with dependent communication types and singleton types for names and capabilities. Technical report 2002-3, Department of Computing and Information Sciences, Kansas State University, December 2002.

[38] Ian Westmacott, J. B. Wells, Robert Muller, and Torben Amtoft. A mechanical verification of region inference: using an automatic theorem prover to verify a type-based program transformation. Submitted for publication, 2002.

[39] Torben Amtoft. Causal type system for ambient movements. Submitted for publication, October 2001. Technical report 2002-04, Department of Computing and Information Sciences, Kansas State University, 2002.

[40] Torben Amtoft. Partial evaluation for constraint-based program analyses. BRICS Technical Report BRICS-RS-99-45, DAIMI, University of Aarhus, Denmark, 1999.

[41] Torben Amtoft. Strictness types: An inference algorithm and an application. Technical Report PB-448, DAIMI, University of Aarhus, Denmark, August 1993.

[42] Torben Amtoft. *Sharing of Computations*. PhD thesis, DAIMI, University of Aarhus, Denmark, 1993. Technical report PB-453.

[43] Torben Amtoft and Jesper Larsson Träff. Memoization and its use in lazy and incremental program generation. Master's thesis, DIKU, University of Copenhagen, Denmark, August 1989. No. 89-8-1.

Invited Talks and Research Presentations

Conference/workshop talks

A theory of slicing for probabilistic control flow graphs. FoSSaCS'16, Eindhoven, Netherlands, April 2016.

Precise and automated contract-based reasoning for verification and certification of information flow properties of programs with arrays. ESOP'10, Paphos, Cyprus, March 2010.

From Generic to Specific: Off-line Optimization for a General Constraint solver. GPCE'08, Nashville, Tennessee, October 2008.

Verification Condition Generation for Conditional Information Flow. FMSE'07, Fairfax, Virginia, November 2007.

A New Foundation for Control-Dependence and Slicing for Modern Program Structures. ESOP'05, Edinburgh, Scotland, April 2005.

Information Flow Analysis in Logical Form. SAS'04, Verona, Italy, August 2004.

What are polymorphically-typed ambients? ESOP'01, Genova, Italy, April 2001.

Faithful translations between polyvariant flows and polymorphic types. ESOP'00, Berlin, Germany, March 2000.

Local type reconstruction by means of symbolic fixed point iteration. ESOP'94, Edinburgh, Scotland, April 1994.

Minimal thunkification. WSA'93, Padova, Italy, September 1993.

Unfold/fold transformations preserving termination properties. PLILP'92, Leuven, Belgium, August 1992.

Properties of unfolding-based meta-level systems. PEPM'91, New Haven, Connecticut, June 1991.

Invited talks (since 2000)

Program Slicing and its Correctness (History and Recent Trends). Midwest Verification Days 2014, University of Missouri in Columbia, October 4th, 2014.

Correctness of Slicing Finite State Machines. Harvard University, August 19th, 2013.

Hoare-like Logics for Verifying and Inferring Conditional Information Flow. 19th CREST Open Workshop on Interference and Dependence, University College London, May 1st, 2012.

Slicing for Modern Program Structures: a Theory for Eliminating Irrelevant Loops. King's College London, July 22nd, 2009; Technical University of Denmark, June 2nd, 2008.

Verification Condition Generation for Conditional Information Flow. Northeastern University, May 30th, 2007.

A Logic for Information Flow in Object-Oriented Programs. Technical University of Denmark, May 23rd, 2006; University of Copenhagen, May 22nd, 2006.

Information Flow Analysis in Logical Form. University of Copenhagen, August 5th, 2004; Open Source Quality Project Retreat (Santa Cruz, California), May 14th, 2004.

The Semantic Soundness of a Type System for Interprocedural Register Allocation and Constructor Flattening. Boston University, June 2nd, 2003; Northeastern University, May 28th, 2003. (These presentations were assisted by Robert Muller.)

Causal Type System for Ambient Movements. Boston University, December

16th, 2002.

Causal Type Systems for Ambients. Technical University of Denmark, August 17th, 2001.

What are Polymorphically Typed Ambients? Kansas State University (interview talk), April 26th, 2001; Universita Ca' Foscari di Venezia, April 10th, 2001; NEPLS at Brown University (Providence, RI), December 7, 2000.

Faithful Translations between Polyvariant Flows and Polymorphic Types. University of Copenhagen, April 3rd, 2000; NJPLS (New Jersey Programming Language Seminar), September 1, 1999.

Behaviour Analysis for Validating Communication Patterns. Northeastern University, March 15th, 2000.

Professional Contributions

Refereeing

I served on the program committees for

- FoSSaCS 2012 (15th International Conference on Foundations of Software Science and Computation Structures), held as part of ETAPS in Tallinn, Estonia, Spring 2012. (LNCS volume 7213.)
- IFL'11 (23rd Symposium on Implementation and Application of Functional Languages), held in Lawrence, Kansas, Fall 2011.
- ESOP'09 (European Symposium on Programming), held as part of ETAPS 2009 in York, United Kingdom, Spring 2009. (LNCS volume 5502.)
- ICFP'06 (The 11th ACM SIGPLAN International Conference on Functional Programming), held in Portland, Oregon, September 2006.
- ESOP'04 (European Symposium on Programming), held as part of ETAPS in Barcelona, Spain, Spring 2004. (LNCS volume 2986.)
- ITRS'04 (Workshop on Intersection Types and Related Systems), held co-located with LICS and ICALP in Turku, Finland, July 2004.
- PADO-II (Programs as Data Objects), a symposium held with MFPS 2001 in Aarhus, Denmark, May 2001. (LNCS volume 2053.)

I have served as a reviewer on 41 journal submissions: 8 times for *ACM Transactions on Programming Languages and Systems*; 5 times for *Higher-Order and Symbolic Computation*; 5 times for *Information and Computation*; 4 times for *Information Processing Letters*; 4 times for *Journal of Functional Programming*; 3 times for *Theoretical Computer Science*; 2 times for *Formal Aspects of Computing*; 1 time for each of *ACM Computing Surveys*, ACM Transactions of Computational Logic (ToCL), *Computer Languages*, *IEEE Transactions on Software Engineering*, *International Journal of Foundations of Computer Science*, *International Journal on Software Tools for Technology Transfer (STTT)*, *Journal of Computer Science and Technology*, *Journal of Systems and Software*, *Logical Methods in Computer Science*, *Studia Logica*.

On numerous occasions, I have reviewed conference submissions on request from program committee members. The conferences include POPL (Principles of Programming Languages), 15 submissions; SAS (Static Analysis Symposium), 14 submissions; ICFP (International Conference on Functional Programming), 12 submissions; ESOP (European Symposium on Programming), 9 submissions; CONCUR (Conference on Concurrency Theory), 7 submissions; ICALP (International Colloquium on Automata, Languages, and Programming), 5 submissions; PEPM (Partial Evaluation and Semantics-based Program Manipulation), 5 submissions.

Invited Seminar and Conference Participation

Workshop-fest in Honor of Neil D. Jones, Copenhagen, Denmark, August 25-26, 2007.

First International Workshop on Programming Language Interference and Dependence (PLID). Verona, Italy, August 25, 2004.

Dagstuhl Seminar 03411 on Language-Based Security. October 5–10, 2003.

Advanced Course on the *Principles of Program Analysis*. Dagstuhl Event No 98451, November 9–13, 1998.

Professional Development

I have attended numerous international conferences, for example ETAPS (European Joint Conferences on Theory and Practice of Software) 7 times

(2000, 2001, 2005, 2009, 2010, 2012, 2016), and POPL (Principles of Programming Languages) 5 times (1995, 1997, 1999, 2000, 2001).

In 2017, I attended in Portugal a summer school on probabilistic programming which was the 1st edition of the "School on Foundations of Programming and Software systems" which is jointly funded by EATCS, ETAPS, ACM SIGLOG, and ACM SIGPLAN.

Scientific visits include: Mark Harman's group in London, at King's College (July 2009) and University College (May 2012 and August 2013), Bob Muller at Boston College (May/June 2003), Michele Bugliesi at Universita' "Ca Foscari", Venezia (April 2001), Lone Leth & Bent Thomsen at ECRC Munich (April 1996) and ICL London (November 1996), Mitch Wand at Northeastern University, Boston (January 1995).

Outreach includes team visits (June 2006 & June 2007 & October 2007) to Rockwell Collins' Advanced Technology Center in Cedar Rapids, Iowa, who has funded research done by our team (led by John Hatcliff).

Personal Record

Nationality citizen of Denmark; citizen (since January 2017) of the U.S.

Languages Danish; English; a little (and rusty) German, Russian, French.

References can be obtained from, e.g., Flemming Nielson, Hanne Riis Nielson, Olivier Danvy, Assaf Kfoury, J.B. Wells, David Schmidt, Anindya Banerjee, John Hatcliff.

Appendix: Detailed Summary of Research

Below is a description of my research contributions, categorized by topic; a chronological ordering of the topics has been attempted.

Memoization and theory of computation. [29] investigates the meaning and practical use of the 2nd recursion theorems by Kleene and Rogers. My M.Sc. thesis [43] is about how to improve efficiency of program execution by means of a generalized form of memoization, resembling partial

evaluation; as reported in [10], Cook's ingenious linear-time simulation of 2DPDAs can be thought of as an instance of this technique.

Models for program optimization. The main purpose of my Ph.D. thesis [42] is to develop a model enabling one to reason about various techniques for program optimization, in particular wrt. *speedup* and *correctness*. Concerning speedup, some of the results are presented in [28]; in particular the reasons why a program transformation may yield *more than* a constant speedup are factored out. Concerning correctness, some results (generalizing previous approaches from the literature) about preservation of termination properties for a logic language are presented in [27].

Specification of analysis and transformation. In [41], strictness analysis is formulated in terms of type inference. A type reconstruction algorithm is presented, and the strictness information is used to avoid some superfluous "thunkifications" when translating from call-by-name into call-by-value. Of particular interest is the proof technique: the correctness of the translation is proved *simultaneously* with the correctness of the analysis. The part concerning type reconstruction is published in [25]; the part concerning translation is published in [26].

Effect analysis for concurrent systems. [9] develops a sound and complete type and behavior reconstruction algorithm for a fragment of Concurrent ML (CML), the starting point being the inference system presented by Hanne Riis Nielson and Flemming Nielson at POPL'94. The algorithm returns a set of constraints; and we show how to solve these in the monomorphic case (but not in general).

The monograph [1] gives an overview over type and effect systems, and then (improving upon the results of [31], [32] and [33]) develops an annotated type and effect system for a fragment of CML; the system uses constraints on the left hand side of the turn-stile and integrates Hindley-Milner polymorphism, subtyping, and effects. We show that the system is semantically sound; and develop a reconstruction algorithm that is sound and also complete. This algorithm has been used as the basis of a prototype implementation, available for experimentation on the WWW. [8] contains a description of the system, illustrated by several examples, as well as a brief account of the underlying theory. [24] shows that the system greatly assists in validating a number of safety properties for "realistic" concurrent systems.

Applications of partial evaluation. [40] reports on experiments investigating whether control flow analysis can be optimized by partially evaluating the analyzer. So far the results have been negative, except that the residual program pinpointed a serious source of inefficiency, leading to the (re)invention of an incremental version of the analyzer.

[30] exposes how one by partial evaluation of a single generic string matching algorithm can achieve the effect of the Knuth & Morris & Pratt string matcher, as well as the effect of (several variants of) the Boyer & Moore string matcher. This has been known for at least a decade, when similar results were made public by the authors (together and independently); the primary goal of this paper is thus to summarize the findings and put them into perspective.

Frameworks for polyvariant analysis. [23] demonstrates that there is a close relationship between polyvariant flow analyses and type systems with finitary polymorphism. We present a flow logic, based on the general approach of Nielson & Nielson and augmented with ideas from Palsberg & Pavlopoulou, and also present a type system employing union and intersection types; both these systems satisfy a subject reduction property. We then provide translations between types and flows that are "faithful" in that they act as the identity on "canonical" elements, and otherwise canonicalize.

Type systems for the ambient calculus. [22, 7] and [39, 3] consider the Ambient Calculus, proposed by Cardelli and Gordon as a formal framework to study issues of mobility and migrant code, and develop type systems for the calculus. These systems employ a notion of causality in that processes are assigned "behaviors", where a behavior is essentially a regular set of traces. Thus type checking (of fully annotated processes) is decidable, using techniques borrowed from finite automata theory. (Under certain restrictions, type inference is also possible.)

In [22], the focus is on extending the ambient calculus so as to allow a more natural, yet safe, style of programming. This is done by embedding a functional language, and by designing the type system to smoothly integrate several kinds of "polymorphism": *(i)* the well-investigated notion of subtyping; *(ii)* "arity polymorphism", allowing the same ambient to hold several topics of conversation *simultaneously*; and *(iii)* "orderly communication", allowing the same ambient to hold several topics of conversation *consecutively*. A subject reduction property ensures that communicating

subprocesses agree on their "topic of conversation". As "orderly communication" is the main technical innovation of the above, the journal paper [7] concentrates on this feature only.

In [39], summarized and put into perspective in [3], the focus is on security in that the type system is parameterized by a set of security constraints: static ones expressing where a given ambient may reside, and dynamic ones expressing where a given ambient may be dissolved. A subject reduction property then guarantees that a well-typed process never violates these constraints. It is argued that the presence of causality significantly increases the precision of the analysis and compensates for the lack of "co-capabilities" (an otherwise increasingly popular extension to the ambient calculus).

The goal of [37], and the further development in [20], is to provide type polymorphism of the kind that is usually present in polymorphic type systems for the λ-calculus, thereby allowing mobile agents to follow non-predetermined paths and to carry non-predetermined types of data from location to location. This is achieved by letting the type of an ambient process P give an upper bound on the possible ambient nesting shapes of any process P' to which P can evolve. Because these shapes can depend on which capabilities and names are actually communicated, the types support this with explicit dependencies on communication. The type of an ambient name may thus depend on where the ambient has traveled, whereas in previous type systems for ambient calculi, there is a global assignment of types to ambient names.

Type systems for register allocation. In [21], we design for a compiler intermediate language an annotated type system supporting interprocedural register allocation and the representation of tuples and variants directly in the register file.

Information Flow Analysis. In [19], we specify an information flow analysis for a simple imperative language, using a Hoare logic. The logic facilitates static checking of a larger class of programs than can be checked by extant type-based approaches in which a program is deemed insecure when it contains an insecure subprogram. The logic is based on an abstract interpretation of program traces that makes independence between program variables explicit. Unlike other, more precise, approaches based on Hoare logic, our approach does not require a theorem prover to generate invariants. We demonstrate the modularity of our approach by showing that a frame rule holds in our logic.

In [6], we extend the results of [19] to handle also nontermination sensitive information flow analysis, and show how the logic gives rise to a (provably correct) algorithm for forward slicing.

In [17], we modify the logic of [19] to handle object-oriented languages; to reason about aliasing, ubiquitous in such languages, the information flow logic is built on top of a logic for abstract locations (just as any analysis for higher order functional programs is built on top of a control-flow analysis). The logic enjoys "small" specifications, so as to facilitate modular reasoning; these can be combined by a frame rule. Under certain assumptions, it is possible to compute "strongest postconditions". Our language permits programmer assertions, in the style of ESC/Java.

In [16], we present an alternative approach to information flow analysis of sequential heap manipulating programs. We use "object flow invariants" to express the information flow properties an object must satisfy; such an invariant may be temporarily violated while an object is being updated but must be restored at the end (when the "scope" of the object is closed). Hence there is no need for explicit reasoning about aliasing. In general, information flow properties are expressed using assertions that are conditional in that they depend on standard Hoare assertions being satisfied. We define an algorithm `VCgen` which from a program and its desired postcondition generates a precondition which is strong enough to establish the postcondition. `VCgen` must be supplied with object flow invariants as well as with flow invariants for loops; if these invariants are not strong enough then the verification conditions generated by `VCgen` cannot be satisfied. The current analysis is intraprocedural.

In [15], we employ the techniques of [16], extended to an interprocedural setting, to enhance the SPARK information flow annotation language with conditional information flow contracts. Unlike [16], we now have a method for automatically inferring loop flow invariants; therefore contracts can be compositionally checked and inferred (the SPARK subset of Ada deliberately omits constructs that are difficult to reason about, such as heap objects; hence we do not need to worry about inferring flow invariants for such). We report on the use of this framework for a collection of SPARK examples; our experiments are based on an implementation that allows various degrees of assertion simplification.

In [13], we extend [15] so as to enable precise compositional specification of information flow in programs with arrays. This has substantial practical impact since SPARK does not allow dynamic allocation of memory and

hence, to implement complex data structures, makes heavy use of arrays. These have previously been treated as indivisible entities; flows that involve only particular locations of an array had to be abstracted into flows on the whole array. The main technical novelty of [13] is an algorithm for inferring universally quantified contracts for `for` loops. We demonstrate the expressiveness of the enhanced contracts, and the effectiveness of the automated verification algorithm, on realistic embedded applications.

In [12], we extend the framework of [13] so that the algorithm for verifying source code compliance to an information flow contract emits formal certificates of correctness, to be checked by the Coq proof assistant. For a core subset of the source language, we have proved in Coq that if a program can be given a certificate which is well-typed then the program does indeed satisfy the semantic information flow properties specified by the certificate.

Slicing. In [18], we examine the notion of *control dependence*, underlying many program analysis techniques such as slicing. We argue that existing definitions are difficult to apply seamlessly to modern program structures which make substantial use of exception processing and increasingly support reactive systems designed to run indefinitely; we repair on that by developing definitions that apply also to control flow graphs without end nodes (or with more than one end node) and which conservatively extend classic definitions. For one of the new definitions, we show the correctness of the induced slicing algorithm, wrt. a correctness criterion based on weak bisimulation. Algorithms for computing the new control dependences form the basis of a publicly available program slicer that has been implemented for full Java.

In [5], we extend [18] so as to handle control flow graphs without end nodes also if they are *irreducible*. This requires a new notion of control-based dependence, called "order dependence". A detailed correctness proof is given for the slicing induced by the modified definitions.

[4] provides a feature missing in [5]: the foundation of an approach to slicing which allows for the elimination of loops that do not affect the values of relevant variables, and thereby is more likely to generate slices of manageable size. The corresponding correctness criterion is based on "weak simulation", implying that the observational behavior of the original program is a *prefix* of the behavior of the sliced program. A crisp correctness proof shows that for slicing to satisfy this correctness property, even wrt. control flow graphs that are irreducible or have no end nodes, it is sufficient that the given slice

set is closed under (data dependence and) "weak order dependence", one of the new dependencies proposed in [5].

[2] shows that for a control-flow graph where all nodes are reachable from each other, the abovementioned notion of weak order dependence can be expressed in terms of traditional control dependence where one node has been converted into an end node.

[36] provides a foundation for slicing in a *non-deterministic* setting, addressing two key correctness properties. "Weak correctness", which allows to slice away irrelevant loops, now requires not only that each observable action by the original program can be simulated by the sliced program but also, so as not to increase non-determinism, that each observable action by the sliced program can be simulated by the original program, *unless* the original program gets stuck or it loops, the latter possibility excluded if we consider "strong correctness". To ensure weak/strong correctness, we do not try to invent new suitable control dependence relations but instead simply demand that the slice set (in addition to being closed under data dependence) satisfies the properties of "weak commitment"/"strong commitment" proposed by Danicic et al. We carry out the development in the setting of "extended finite state machines" (EFSMs), and also prove that for each of the properties "weak commitment" and "strong commitment" there exists a *least* set with that property and that least slice set can be computed by a low polynomial algorithm. We conduct extensive experiments with widely-studied benchmark and industrial EFSMs so as to measure the relative sizes of slices produced by our algorithms, and to compare with the results of slicing algorithms that use existing definitions of control dependence.

[11] provides a foundation for slicing in a *probabilistic* setting, where variables may be assigned random values from a given distribution, and undesirable combinations of values may be removed by "observe" statements; in the presence of these features, the standard notions of data and control dependence no longer suffice for semantically correct slicing, as demonstrated by Hur et al. in recent work. We present a theory for slicing probabilistic programs, represented as control flow graphs whose nodes transform probability distributions. We separate the specification of slicing from its implementation: first we develop syntactic conditions that a slice must satisfy; next we prove that any such slice is semantically correct; finally we give an algorithm to compute the least slice. The theory is a non-trivial extension of the recent framework by Danicic et al. that unified previous works on slicing and provided solid semantic foundations to the slicing of a large class of (deter-

ministic) programs. Our correctness results states that the original program and the sliced program have the same final probability distribution, modulo a constant factor so as to allow the removal of "observe" statements that do not introduce any bias in the final distribution. This will be the case if the variables tested by "observe" statements are probabilistically independent of those variables relevant for the final value. To ensure this, a key feature of our syntactic conditions is that they involve two disjoint slices, such that the variables of one are probabilistically independent of the variables of the other.

[35] is an extended version of [11]; the main additional contribution is that we can now slice away certain loops if they are known (through some analysis, or an oracle) to terminate with probability 1. [34] focuses on the semantics for probabilistic control-flow graphs used in [35], and in particular states its adequacy with respect to the "classical" semantics for structured probabilistic programs.