VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kansliets noteringar Kod	Dnr
2013-2993-109043-9	

2013
Grants for distinguished professors

Area of science
Natural and Engineering Sciences
Announced grants
Grants for distinguished Professors 2013
Total amount for which applied (kSEK)
2014 2015 2016 2017 2018

APPLICANT

Name(Last name, First name)
Jansson, Patrik
Date of birth
720311-7515
Gender
Male
Email address
patrikj@chalmers.se
Academic title
Professor
Position
Biträdande professor
Phone
031-7725415
Doctoral degree awarded (yyyy-mm-dd)
2000-06-19

ADMINISTRATING ORGANISATION

Administrating Organisation
Chalmers tekniska högskola

WORKING ADDRESS

University/corresponding, Department, Section/Unit, Address, etc.
Chalmers tekniska högskola
Institutionen för data-och informationsteknik
Programvaruteknik

41296 Göteborg, Sweden

PROPOSED HOST ORGANISATION

University/corresponding, Department, Section/Unit, Address, etc.
Chalmers tekniska högskola
Institutionen för data-och informationsteknik
Programvaruteknik

41296 Göteborg, Sweden

UNIVERSITY, FACULTY AND DEPARTMENT FOR DOCTORAL EXAM

DESCRIPTIVE DATA

Project title, Swedish (max 200 char)
Starkt typade bibliotek för program och bevis

Project title, English (max 200 char)
Strongly Typed Libraries for Programs and Proofs

Abstract (max 1500 char)
Our long-term goal is to create systems (theories, programming languages, libraries and tools) which make it easy to develop reusable software components with matching specifications. In this research project, the main focus is on libraries. Strongly-typed programming languages allow to express functional specifications as types. Checking the types of a program then means checking it against its specification. Within such powerful programming languages, libraries are not only building blocks of programs, but also of proofs. We believe that such libraries will eventually become the main means of developing programs, and because

they come with strong types, the programs built using the library will come with strong properties that will make the whole easy to prove correct. The production of such libraries will also inform the design of future strongly-typed programming languages. In the recent years, strongly-typed programming languages have started to become usable, but remain confined to a small niche. Our libraries will make them a viable solution for a broader range of applications, bringing higher guarantees of correctness to a wider user base. To check the applicability of our libraries, we will apply them to classical problems of computer programming, such as certain divide-and-conquer algorithms or optimisation problems, as well as to the construction of tools supporting dependently-typed programming themselves.

Abstract language
English
Keywords
Software Technology, Functional Programming, Dependent Types, Program Verification, Generic Programming
Review panel
NT-RP
Classification codes (SCB) in order of priority
10201, 10205, 10103
Aspects

ENCLOSED APPENDICES

A, B, C, S

APPLIED FUNDING: THIS APPLICATION

Funding period (planned start and end date)
2014-01-01 -- 2023-12-31
The amount is predetermined by the Swedish Research Council.

Popularscience heading and description (max 4500 char)
En viktig gren av forskningen inom datavetenskap handlar om att utveckla system (programspråk, verktyg, programbibliotek, teorier) som gör det enkelt att konstruera programvara som är korrekt och återanvändbar. Detta projekt siktar på att utnyttja funktionella programspråk med starka typsystem till att skapa bibliotek av komponenter som kan uttrycka både specifikationer och implementationer som uppfyller dessa. Vi kommer att utnyttja datorstödd interaktiv programutveckling där automatiska verktyg ger snabb återkoppling på vilka delar som inte uppfyller specifikationen.

Den teoretiska möjligheten att uttrycka program och bevis i samma programspråk är känd sedan många år, men det är först nyligen som teknikutvecklingen har medgett att utveckla större programbibliotek på detta sätt. Detta innebär att det finns många spännande grundläggande frågor kvar att utforska och vi avser börja med enkla algoritmer för att sedan steg för steg utforska hur långt det går att komma. Vi arbetar iterativt i tre nivåer för att utveckla komponentbiblioteken. Första nivån är att implementera en lösning på ett visst problem (sökning, optimering eller liknande), nästa nivå är att abstrahera ut gemensamma mönster till programbibliotek och slutligen vill vi utvärdera vilka möjliga förändringar av den underliggande språket som skulle kunna förbättra resultaten. Inom projektet kommer vi att arbeta fram korrekta generiska bibliotek uttryckta i språket Agda. Agda är ett verktyg baserat på typteori och funktionell programmering som möjliggör utveckling av program och specifikationer i samma språk. Utvecklingen av språket har skett (och forskrider

parallellt med biblioteksprojektet) i ett internationellt samarbete (med Japan, Tyskland och England) lett av Chalmers.

På lång sikt kan bevisbart korrekta programbibliotek användas och återanvändas som byggstenar vid all slags programvarukonstruktion. Detta ger allmänt sett mer pålitliga program, och färre buggar. Ett spännande applikationsområde är exekverbara, överblickbara högnivåmodeller för komplexa system. Vi har hittills mest fokuserat på att modellera komplexa system inom dataområdet (logiska ramverk, lingvistik, programspråk, hårdvara) men i samarbetet med Potsdams Institut för Klimatforskning (PIK) har vi börjat arbeta med komplexa system i interaktionen mellan klimat, ekonomi och samhälle. PIK har under flera år arbetat med simuleringar av komplexa system och har under senare år börjat använda funktionell programmering som ett verktyg för att experimentera med och kommunicera de högnivåmodeller som behövs för att överblicka komplexa system. Dessa högnivåmodeller översätts senare i flera steg till effektiv programkod som klarar att köra tunga simuleringar inom rimlig tid. (Dessa simuleringar ger underlag till politiska beslut inom klimatområdet.) PIK tog kontakt med Chalmers för att fördjupa sin kompetens inom högnivåmodellering med hjälp av moderna programspråk (som Haskell och C++) och vi har under åren som gått haft flera kontakter där starkt typade bibliotek för program och bevis har utkristalliserats som det forskningsområde där Chalmers bäst kan komplettera PIK. Samarbetet har lett till ett gemensamt EU-projekt, flera artiklar och bibliotek för program och bevis.

På Chalmers leds projektet av Patrik Jansson (inom gruppen Funktionell Programmering). Jansson har forskat om generisk programmering sedan 1995 i olika konstellationer och det internationella kontaktnätet är mycket starkt. Den lokala forskningsmiljön inom D&IT-institutionen är världsledande även inom flera närliggande områden - automatisk testning (Hughes, Claessen), domänspecifika språk (Sheeran, Claessen), typteori (Coquand), språkteknologi (Ranta).

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2013-2993-109043-9

Name of applicant
Jansson, Patrik

Date of birth
720311-7515

Title of research programme
Strongly Typed Libraries for Programs and Proofs

Appendix A

Research programme

Vetenskapsrådet, Box 1035, SE-101 38 Stockholm, tel. +46 (0)8 546 44 000, vetenskapsradet@vr.se

Strongly Typed Libraries for Programs and Proofs

1 Purpose and aims

Our long term goal is to create systems (theories, programming languages, libraries and tools) which make it easy to develop software components and matching specifications. In this research project, we aim to leverage the power of languages with strong types to create libraries of components which can express functional specifications in a natural way, and, simultaneously, implementations which satisfy those specifications. The ideal we aim for is not merely correct programs, nor even proven correct programs; we want proof done against a specification that is naturally expressed for a domain expert.

Concretely, we aim to identify common patterns in the *specification* of programs, and capture those in libraries. At the same time, the patterns of *implementations* of these specifications will also be captured in the library, such that the development of software will go hand-in-hand with proofs of its functional correctness. As case-studies we will work in three areas: divide-and-conquer algorithms, domain specific modelling and testing tools.

2 Survey of the field

Abstraction. The ability to name and reuse parts of algorithms is one of the cornerstones of computer science. Abstracting out common patterns enables separation of concerns, both in the small (variables, functions) and in the large (modules, libraries). Conversely, lack of abstraction may force the implementation to contain multiple instances of a single pattern. This process of replication is not only tedious, but error-prone, because the risk of software error grows with the size of the program. Hence, one important trend in the evolution of new programming languages is improved support for abstraction—making more and more of the language features programmable. Widely used modern languages are actively gaining abstraction power with Java Generics, C++ Templates, Scheme's composable macros and Haskell meta-programming. But there is a danger lurking—more complex features can increase the risk of bugs. With new abstraction mechanisms we also need new computer-aided sanity checks of the program code.

Types. Types are used in many parts of computer science to keep software from going wrong. Types enable the programmer to keep track of the structure of data and computation in a way that is checkable by the computer itself. They act as contracts between the implementer of a program part and its users. Type-checking at compile time then amounts to proving that properties hold for all executions of the program.

By the Curry–Howard correspondence, type systems closely match up with logics. Type systems for languages with higher-order abstraction, correspond to higher-order logics. A well-know example of a system based on this principle is the Coq proof assistant [1].

Dependently typed programs. Even though type-theory has been used as a logic for decades, it has recently gained popularity as a medium for programming. The flagship of dependently-typed programs is perhaps

Compcert, a C compiler written and verified in Coq [2]. Other applications include imperative programs within Coq, database access with static guarantees and distributed programming with dependent types [3].

Agda. The programming language Agda is based on Haskell and type-theory [4]. Within it, one can express programs, functional specifications as types, and proofs (for example using algebraic reasoning) in a single language. Agda is currently emerging as a lingua-franca of programming with dependent types and its canonical reference, Norell's Thesis [5], has been cited 50 times per year since its publication. The focus of this project is on expressing libraries of correct programs and proofs in the dependently typed functional language of Agda.

Libraries for dependent types. Agda and Coq come with standard libraries that contain useful building blocks to create programs, specifications, and proofs. The Coq library is part of a mature system which has been used in many projects. However, it is mostly applied to proofs rather than programs, because the Coq system is mostly intended as a proof assistant rather than a programming language. Even projects which aim to use Coq as a programming platform, such as Ynot and Compcert [2, 6] retain this separation. The same observation applies to the libraries of most systems with dependent types. The Agda standard library (developed mainly by Danielsson), has evolved from common abstractions needed by Agda programmers. It has been applied to several domains, in particular parser combinators [7], Algebra of Programming [J19][1] and Cryptography (see the `DemTech.dk` project). In its current version, the parts of the Agda library aimed at programming are essentially decoupled from the parts aimed at proving. This can be a drawback: the structure of a proof often follows the same structure as the program it refers to, therefore keeping the two separated violates the principle of abstraction described above.

[1] We use [Jnn] to cite papers in Jansson's publication list (appendix C) to avoid duplication.

3 Programme description

Starting from our current experience with libraries for Haskell and Agda, our project will be organised in multiple iterations, each refining the libraries obtained during the previous one. Each iteration will have the following three phases.

1. Development of a proven-correct application in a given domain. We believe that the best way to develop libraries is by abstracting common patterns found in various application domains. In this phase, we will assess the viability of our libraries by applying them to a particular domain.

2. Extraction of common patterns into libraries. Next, we will identify common patterns found in the programs and specifications produced above, and capture them in libraries. At the same time, we will refactor the application and tie each specification pattern to an implementation pattern.

3. Refining the prog. language. In this phase we will assess the strong and weak points of the underlying programming environment we use. We will inform the group in charge of the development of the tool of the possible shortcomings we might identify, and participate in their remedy, if suitable.

3.1 Application areas

We work iteratively on case studies (correct libraries) in three application areas:

Algebra of Parallel Programming: A large class of sequence-processing algorithms can be converted to parallel algorithms if they are monoid homomorphisms. That is, a function $f : A \to B$ can be parallelised if it satisfies the following laws:

$$\begin{aligned} f\ empty_A &= empty_B \\ f\ (a \mathbin{+\!\!+}_A b) &= f\ a \mathbin{+\!\!+}_B f\ b \end{aligned}$$

where *empty* and $+\!\!+$ denote monoidal unit and composition. In some cases, the function is not a monoid homomorphism, but it can be phrased in terms of an auxiliary function, which works on an extended type.
A simple example is word counting, which maps each string to a natural number (the number of white-space separated words in the string). Word counting is not by itself a monoid homomorphism but if we keep the number of full words, plus some information about spacing on either side we can make a homomorphic *helper*.

$$\begin{aligned} &helper &&: String \to (Count, Spa) \\ &countSpaces &&: (Count, Spa) \to \mathbb{N} \\ &... \\ &wordCount &&: String \to \mathbb{N} \\ &wordCount &&= countSpaces \circ helper \end{aligned}$$

The resulting algorithm works for any tree-like partitioning of the string into chunks and can thus be made fast by using many processors. In this project we want to develop a library for specifying, implementing and proving correctness of divide-and-conquer algorithms. Initial results by our group show that this works even for something as sequential-looking as parsing [8].

Domain specific modelling: What good is proof of correctness is if no-one understands the specification? We take the stance that specifications must be readily understood by domain experts, and therefore it is important for computer-scientists to work with the domain specific concepts. We have done so in the past, in the domain of vulnerability for climate impact [J17], grammars for language processing [J13], and mathematical economics [J9]. In addition to domain knowledge this requires support for higher-order constructions like monads, functors and natural transformations. In this part we will focus on optimisation and dynamical systems. In particular, we will continue the work on economics and develop libraries of specifications for agent-based modelling. We will also work on improving language support to present the specifications in a way accessible to the domain experts.

Testing Tools: Property-based testing tools have proved useful to improve the confidence in program correctness. As it is well known, testing cannot show the absence of bugs, only their presence. But is it possible to quantify the confidence gained by running a test suite? We aim to give a positive answer to the question. A first step in this direction is to specify the set of inputs covered by a test-suite. In this project we will focus on large abstract syntax tree (AST) types typically used in compilers, and aim at supporting interesting subsets like well-typed terms or balanced trees (expressible as inductive families in Agda).
In a recent paper [J11] we presented a theory for specifying, and a generic Haskell library for efficiently enumerating, the terms of complex AST-types. The primary application is property-based testing, where it is used to

define both random sampling (QuickCheck generators) and exhaustive enumeration. In this project we will port this library and its specification to Agda and extend it towards inductive families. Our hypothesis is that, compared to QuickCheck, the more algebraic enumeration approach will be easier to specify and prove correct.

4 Preliminary findings

We have published initial results in all the three suggested iteration phases and three application areas as indicated below.

Proven-correct applications: We have worked on correct applications in Haskell [9] [J5] and supporting theory [J21]. We have also worked on applications to climate impact research and economic modelling directly in Agda [J9, J10].

Patterns into libraries: We have developed, implemented and compared libraries of generic functions [10, 11] [J16]. Most of this has been done in Haskell, but it has become clear that the natural setting for generic programming is dependent types. We have also worked on libraries for parsing [12] [J13], testing [J11, J12] and the above mentioned applications to climate and economy.

Refinement of prog. languages: We have designed a generic programming language extension (PolyP [J1]) for Haskell, and we are actively involved in the design of the Agda language [J30] [13]. We have also been involved in the development of the Concepts feature aimed at improving future versions of C++ and Haskell [J7, J18].

Algebra of Parallel Programming: We have worked actively on implementing programs and proofs in the Algebra of Programming [J2, J19] tradition. Recent work includes an efficient sparse matrix based algorithm for parallel parsing and its proof [8].

Domain Specific Modelling: Dependent type theory is rich enough to express that a program satisfies a functional specification, but there is no *a-priori* method to derive a program once the specification-as-type is written. On the other hand, Bird & de Moor [14] give a general methodology to derive Haskell programs from specifications, via algebraic reasoning. Despite the strong emphasis on correctness, their specifications and proofs are not expressed in a formally checkable way. In [J8] we have shown how to encode program derivation in the style of Bird and de Moor, in Agda. A program is coupled with an algebraic derivation from a specification, whose correctness is guaranteed by the type system. In this project we want to go further in this direction and develop useful libraries of programs and proofs with corresponding types and theorems.

Testing and verification: We have explored the tension between testing and proving of higher-order properties [J20, J10], developed a technique for drastically reducing the number of tests required for polymorphic properties [J15], developed a library for specifying and testing class laws [J12] and a library for enumerating test cases [J11].

We have also started a very promising line of work on *parametricity theory*: in [J14, J6] we have explained how the classic results from "Theorems for free!" [15] can be extended to dependently typed languages to give "Proofs for free". By combining our results from polymorphic testing with our later results on parametricity we aim at a generic library for testing polymorphic properties in Agda.

5 Significance

Effective production of correct software is a problem which remains unsolved, and is of great economic significance. By using dependently-typed languages, this project aims to reduce the potential for errors by developing the specification of a system together with its implementation, and keeping them synchronised throughout the lifetime of the system. An advantage of this approach is that the skills required to construct programs are directly applicable to understanding the specifications. This also means that the libraries become self-documenting.

Software libraries have long been recognised as vehicles for increased software productivity. First, they capture domain knowledge in terms of software solutions to the problems that a user wants to solve. Second, they add a layer of abstraction to the underlying computation, which allows developers to write software in terms closer to their problem domain and usually results in improved quality and robustness. We aim to go beyond state-of-the-art when it comes to expressivity of libraries for programming with dependent types, which is a relatively unexplored niche. By doing so, we hope to improve the software technology field in general, as these libraries should serve as examples of good design for other applications.

The scientific contributions to the computer science area will be in the form of open source software prototypes, doctoral training, conference and journal papers and talks (on the techniques used for the libraries as well as on the language improvements). We also hope to help the wider research community by contributing strongly typed libraries for increasingly correct scientific computing.

References

[1] The Coq development team. The Coq proof assistant, 2010.

[2] Leroy. Formal verification of a realistic compiler. *Comm. of the ACM*, 52(7):107–115, 2009.

[3] Swamy et al. Secure distributed programming with value-dependent types. In *ICFP 2011*, p. 266.

[4] Martin-Löf. *Intuitionistic type theory*. Bibliopolis, 1984.

[5] Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers, 2007.

[6] Chlipala et al. Effective interactive proofs for higher-order imperative programs. In *ICFP 2009*, p. 79. ACM, 2009.

[7] Danielsson. Total parser combinators. In *ICFP 2010*, p. 285. ACM, 2010.

[8] Bernardy & Claessen. Efficient divide-and-conquer parsing of practical context-free languages. To appear in *ICFP 2013*.

[9] Danielsson & Jansson. Chasing bottoms, a case study in program verification. In *MPC 2004*, *LNCS 3125*, p. 85. Springer.

[10] Norell & Jansson. Polytypic programming in Haskell. In *IFL 2003*, *LNCS 3145*, p. 168. Springer, 2004.

[11] Jansson & Jeuring. Functional pearl: Polytypic unification. *J. Funct. Program.*, 8(5):527–536, 1998.

[12] Bernardy. Lazy functional incremental parsing. In *Haskell 2009*, p. 49. ACM.

[13] Bernardy & Moulin. Type-theory in color. To appear in *ICFP 2013*.

[14] Bird & de Moor. *Algebra of Programming*, volume 100 of *International Series in Computer Science*. Prentice-Hall, 1997.

[15] Wadler. Theorems for free! In *Proc. of FPCA 1989*, p. 347. ACM, 1989.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2013-2993-109043-9

Name of applicant
Jansson, Patrik

Date of birth
720311-7515

Title of research programme
Strongly Typed Libraries for Programs and Proofs

Appendix B

Curriculum vitae

Vetenskapsrådet, Box 1035, SE-101 38 Stockholm, tel. +46 (0)8 546 44 000, vetenskapsradet@vr.se

Curriculum Vitæ: Patrik Jansson, 1972-03-11

1. Higher education degree:

1995: BSc+MSc degrees in Engineering Physics + Engineering Mathematics from Chalmers, Sweden. I graduated almost two years before schedule as the best student of my year.

2. Doctoral degree:

2000: Ph.D. degree in Computer Science from Chalmers, Sweden, on *Functional Polytypic Programming*, Advisor: Johan Jeuring.

3. Present position:

2011–now: Professor, Chalmers. Research 50% (2013).

4. Previous Employment and Education:

1998, 1998, 2001: Research visits (2 + 2 + 3 months) to Northeastern University, Boston, USA; Oxford University Computing Lab, UK; Dept. of Computer Science, Yale, USA.

2000–2001: PostDoc, Chalmers.

2001–2004: Assistant Professor in Computer Science, Chalmers.

2004–2011: Associate Professor (docent), Chalmers.

5. Supervision, doctorates and postdocs:

I was PhD advisor of Ulf Norell (PhD 2007), Nils Anders Danielsson (PhD 2007) and Jean-Philippe Bernardy (PhD 2011). I worked on generic programs and proofs with Norell, on program correctness through types with Danielsson and parametricity for dependent types & testing with Bernardy. All three are still in academia.

I have also supervised four PostDocs: Grégoire Hamon and Andreas Abel (2003–2005), Jean-Philippe Bernardy (2011–2012) and now Cezar Ionescu (2013–).

I currently supervise the PhD student Jonas Duregård (Lic. Dec. 2012). I am also examiner (but not supervisor) of three other PhD students: Ramona Enache, Dan Rosén and Anton Ekblad.

I have been a member of the evaluation committee of three PhD defences at Chalmers (T. Gedell, Computer Science (2008), M. Zalewski, Computer Science (2008), H. Johansson, Physics (2010)). I have also supervised over 20 MSc and BSc project students.

6. National or international distinctions and awards:

National and international grants:

1997–2004: Obtained travel grants (**277k SEK** in total) from several private foundations

2003–2005: Co-applicant on *Cover — Combining Verification Methods in Software Development* funded with **8M SEK** by the Swedish Foundation for Strategic Research.

2003–2005: Main applicant on the project *Generic Functional Programs and Proofs* funded with **1.8M SEK** by VR.

2009–2012: Co-applicant on "*Software Design and Verification using Domain Specific Languages*" funded with **11M SEK** by the Swedish Science Council (VR, multi-project grant in strategic Information and Communication Technology).

2010–2013: Co-applicant and work-package leader in the Coordination Action "*Global Systems Dynamics and Policy*" (GSDP) funded with **1.3M EUR** by the EU (ICT-2009.8.0 Future and Emerging Technology Open call).

2011–2016: Co-applicant on "*RAW FP: Productivity and Performance through Resource Aware Functional Programming*" funded with **25M SEK** by the Swedish Foundation for Strategic Research.

2011–2013: Main applicant on the project *Strongly Typed Libraries for Programs and Proofs* funded with **2.4M SEK** by VR.

Commissions of trust:

1998: Organiser of the first Workshop on Generic Programming (WGP), Marstrand. The workshop has later grown into an official Association of Computing Machinery Special Interest Group on Programming LANguages (ACM SIGPLAN) workshop and I've been PC member (2000), PC chair (2009), Steering Committee member (2009–2011) and Chair of the Steering Committee (2011–2012).

2001: PC member of the ACM SIGPLAN Haskell Workshop.

2008–present: Elected member of IFIP (International Federation for Information Processing) Working Group 2.1 on "Algorithmic Languages and Calculi".

2009–2011: Elected member of the faculty senate, Chalmers.

2009–2011: Member of the Steering group of Workshop on Generic Programming.

2009: PC Chair for Workshop on Generic Programming

2011: Organiser of a workshop on "Domain Specific Languages for Economical and Environmental Modelling" in Marstrand as part of EU coordination action GSDP.

2011–2012: Steering Committee Chair of Workshop on Generic Programming.

2012: Workshops chair of the 17th ACM SIGPLAN International Conference on Functional Programming Programming (ICFP 2012).

2012: Organised a workshop on "Computer Science meets Global Systems Science" as part of the First Open Global Systems Science Conference in Brussels.

2012: Co-organised a workshop on "Models and Narratives in Global Systems Science" with Ilan Chabay as part of the First Open Global Systems Science Conference in Brussels.

2013: Organised the "Global Systems Science 2013: Models and Data" workshop in Brussels with Mario Rasetti, Michael Resch and Ralph Dum.

2013: Workshops chair of the 18th ACM SIGPLAN International Conference on Functional Programming (ICFP 2013).

2013: Organised a workshop on "Formal Languages and Integrated Problem Solving procedures in Global Systems Science", Brussels.

2013: Editor for the orientation paper "GSS: Towards a Research Program for Global Systems Science" (with C. Jaeger, S. van der Leeuw, M. Resch and J. D. Tàbara) as part of the effort to define a research area within the upcoming EU framework programme Horizon 2020.

I have been reviewer for EU grants (2011, 2012), Journal of Functional Programming, Science of Computer Programming, Principles of Programming Languages, International Conference on Functional Programming, Symposium on Implementation and Application of Functional Languages and several other journals and conferences.

Awards and prices:

1991: Winner of the Swedish National Physics Olympiad

1991: Represented Sweden in the International Physics Olympiads.

1991: Represented Sweden in the International Mathematics Olympiad.

1996: Received the John Ericsson medal for outstanding scholarship, Chalmers

7. Leadership experience:

2002–2008: Member of the steering group of the department.

2002–2005: Director of Studies for the BSc and MSc education at the Computer Science department.

2005–2008: Vice head of the Computer Science and Engineering dept. responsible for the BSc and MSc education.

2008–2010: Deputy project leader of the IMPACT project at Chalmers ("Development of Chalmers' New Master's Programmes", **30M SEK**).

2009: Head of steering group of Chalmers eScience Initiative.

2011–2013: Head of the 5-year education programme in Computer Science and Engineering (Civilingenjör Datateknik, Chalmers).

2013–: Head of the Division of Software Technology, Chalmers and University of Gothenburg (30% of full time / year). Around 40 employees and a yearly turnover of **47M SEK**.

8. Lecturing experience:

BSc level courses: Lecturer on Imperative Programming in Ada (1998, 1997), Programming Languages (2000, 2001), Databases (2000, 2001), Program Verification (2006), and supervisor of BSc theses (2003, 2009).

MSc level courses: Lecturer of Advanced Functional Programming (2010, 2011, 2012, 2013), and supervisor of MSc theses ($>$ 20 students, 2001–)

PhD level courses: Lecturer of Functional Polytypic Programming (2000), Algebra of Programming (2004, 2008), Category Theory and Functional Programming (2010, 2011) and supervisor of four PhD students (2004–).

9. Future goal:

Just as the transition from spoken language to written language boosted the ability for mankind to accumulate and transmit knowledge from generation to generation the ongoing transition from informal language to formal code promises a giant leap in computer aided knowledge management and inference in all sciences. I want to contribute to this development through pushing for formalisation, automation and verification of research results nationally and internationally starting from the corner I know well and expanding step by step to neighbouring areas. This will require resources well beyond what the Distinguished Professor grant can offer, but I have already worked for three years on informal "lobbying" in the EU machinery for what we now call "Global Systems Science" with the aim of making significant funding available as part of Horizon 2020.

Research in software technology is also cruical for the success of Swedish industry, as shown by the recent report by Swedish ICT: "Innovation enabled by Information and Communication Technologies". I would like to help in lifting this innovation and research agenda to the European level to secure priority support from the European commission for research and development in software technology.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2013-2993-109043-9

Name of applicant
Jansson, Patrik

Date of birth
720311-7515

Title of research programme
Strongly Typed Libraries for Programs and Proofs

Appendix C

List of publications

Vetenskapsrådet, Box 1035, SE-101 38 Stockholm, tel. +46 (0)8 546 44 000, vetenskapsradet@vr.se

Selected Publications: Patrik Jansson

Note to non computer scientists Conference articles in computer science are peer reviewed full articles — not 1–2 page abstracts, and are the normal form of refereed publication. The top conferences in each subfield (like *POPL* and *ICFP* below) typically have the highest impact factor within that field, higher even than any journal.
The publications most relevant for this research project are marked with an arrow (⇒).

0. Most cited publications (Google Scholar, 2013-09-03)

Jansson's Hirsch-index is **17**, his total citation count is **1279** and the following papers are the five most cited (not including the papers in the last 8 years).

1. P. Jansson and J. Jeuring. PolyP — a polytypic programming language extension. In *Proc. POPL'97: Principles of Programming Languages*, pages 470–482. ACM Press, 1997. doi:10.1145/263699.263763.
   Number of citations: **314**.

2. R. Backhouse, P. Jansson, J. Jeuring, and L. Meertens. Generic programming: An introduction. In *Advanced Functional Programming*, volume 1608 of *LNCS*, pages 28–115. Springer, 1999. `http://www.cse.chalmers.se/~patrikj/poly/afp98/`.
   Number of citations: **186**.

3. J. Jeuring and P. Jansson. Polytypic programming. In J. Launchbury et al., editors, *Advanced Functional Programming '96*, volume 1129 of *LNCS*, pages 68–114. Springer-Verlag, 1996. `http://www.cse.chalmers.se/~patrikj/poly/AFP96.pdf`.
   Number of citations: **160**.

4. M. Benke, P. Dybjer, and P. Jansson. Universes for generic programs and proofs in dependent type theory. *Nordic Journal of Computing*, 10(4):265–289, 2003. ISSN 1236-6064. `http://dl.acm.org/citation.cfm?id=985801`.
   Number of citations: **59**.

5. ⇒ P. Jansson and J. Jeuring. Polytypic data conversion programs. *Science of Computer Programming*, 43(1):35–75, 2002. doi:10.1016/S0167-6423(01)00020-X.
   Number of citations: **51**.

1. Peer-reviewed journal articles (last 8 years, excluding the above)

6. J.-P. Bernardy, P. Jansson, and R. Paterson. Proofs for free — parametricity for dependent types. *Journal of Functional Programming*, 22(02):107–152, 2012. doi:10.1017/S0956796812000056.
   Number of citations: **12**.

7. J.-P. Bernardy, P. Jansson, M. Zalewski, and S. Schupp. Generic programming with C++ concepts and Haskell type classes — a comparison. *Journal of Functional Programming*, 20(3–4):271–302, 2010. doi:10.1017/S095679681000016X.
   Number of citations: **10**.

8. ⇒ S.-C. Mu, H.-S. Ko, and P. Jansson. Algebra of programming in Agda: dependent types for relational program derivation. *J. Funct. Program.*, 19:545–579, 2009. doi:10.1017/S0956796809007345.
   Number of citations: **8**.

2. Peer-reviewed conference contributions (last 8 years)

9. ⇒ C. Ionescu and P. Jansson. Dependently-typed programming in scientific computing: Examples from economic modelling. In R. Hinze, editor, *24th Symposium on Implementation and Application of Functional Languages (IFL 2012)*, LNCS. Springer-Verlag, 2013. `http://wiki.portal.chalmers.se/cse/pmwiki.php/FP/DTPinSciComp`.
   Number of citations: **0**.

10. C. Ionescu and P. Jansson. Testing versus proving in climate impact research. In *Proc. TYPES 2011*, volume 19 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 41–54, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.TYPES.2011.41.
    Number of citations: **0**.

11. J. Duregård, P. Jansson, and M. Wang. Feat: Functional enumeration of algebraic types. In *Haskell'12*, pages 61–72. ACM, 2012. doi:10.1145/2364506.2364515.
    Number of citations: **5**.

12. J. Jeuring, P. Jansson, and C. Amaral. Testing type class laws. In *Haskell'12*, pages 49–60. ACM, 2012. doi:10.1145/2364506.2364514.
    Number of citations: **0**.

13. J. Duregård and P. Jansson. Embedded parser generators. In *Proceedings of the 4th ACM Symposium on Haskell*, Haskell '11, pages 107–117, New York, NY, USA, 2011. ACM. doi:10.1145/2034675.2034689.
    Number of citations: **5**.

14. ⇒J.-P. Bernardy, P. Jansson, and R. Paterson. Parametricity and dependent types. In *Proc. 15th ACM SIGPLAN international conference on Functional programming*, pages 345–356, Baltimore, Maryland, 2010. ACM. doi:10.1145/1863543.1863592.
    Number of citations: **29**.

15. ⇒ J.-P. Bernardy, P. Jansson, and K. Claessen. Testing polymorphic properties. In A. Gordon, editor, *European Symposium on Programming*, volume 6012 of *Lecture*

Notes in Computer Science, pages 125–144. Springer, 2010. doi: 10.1007/978-3-642-11957-6_8.
Number of citations: **18**.

16. A. Rodriguez, J. Jeuring, P. Jansson, A. Gerdes, O. Kiselyov, and B. C. d. S. Oliveira. Comparing libraries for generic programming in Haskell. In *Haskell'08*, pages 111–122. ACM, 2008. doi:10.1145/1411286.1411301.
Number of citations: **59**.

17. D. Lincke, P. Jansson, M. Zalewski, and C. Ionescu. Generic libraries in C++ with concepts from high-level domain descriptions in Haskell: A DSL for computational vulnerability assessment. In *IFIP Working Conf. on Domain Specific Languages*, volume 5658/2009 of *LNCS*, pages 236–261, 2009. doi: 10.1007/978-3-642-03034-5_12.
Number of citations: **5**.

18. J.-P. Bernardy, P. Jansson, M. Zalewski, S. Schupp, and A. Priesnitz. A comparison of C++ concepts and Haskell type classes. In *Proc. ACM SIGPLAN Workshop on Generic Programming (WGP)*, pages 37–48. ACM, 2008. doi:10.1145/1411318.1411324.
Number of citations: **21**.

19. S.-C. Mu, H.-S. Ko, and P. Jansson. Algebra of programming using dependent types. In *Mathematics of Program Construction*, volume 5133/2008 of *LNCS*, pages 268–283. Springer, 2008. doi: 10.1007/978-3-540-70594-9_15.
Number of citations: **12**.

20. P. Jansson, J. Jeuring, and students of the Utrecht University Generic Programming class. Testing properties of generic functions. In Z. Horvath, editor, *Proceedings of IFL 2006*, volume 4449 of *LNCS*, pages 217–234. Springer-Verlag, 2007. doi: 10.1007/978-3-540-74130-5_13.
Number of citations: **4**.

21. N. A. Danielsson, J. Hughes, P. Jansson, and J. Gibbons. Fast and loose reasoning is morally correct. In *Proc. Principles of Programming Languages*, pages 206–217. ACM Press, 2006. doi:10.1145/1111037.1111056.
Number of citations: **51**.

3. Review articles, book chapters, books

22. C. Niklasson, P. Jansson, and P. Lundgren. IMPACT - establishing the Bologna structure with master's programmes at Chalmers. In *Utvecklingskonferensen 2008, Nätverket Ingenjörsutbildningarna*, 2008.

23. C. Niklasson, P. Lundgren, and P. Jansson. Utvärdering av Chalmers nya mastersprogram - studentsynpunkter. In *Den 2:a Utvecklingskonferensen för Sveriges ingenjörsutbildningar*, pages 49–52, 2009.

24. P. Jansson and S. Schupp, editors. *WGP'09: Proceedings of the 2009 ACM SIGPLAN workshop on Generic programming*, 2009. ACM. ISBN 978-1-60558-510-9.

25. C. Niklasson and P. Jansson. Pedagogical development of master's programmes for the Bologna structure at Chalmers - IMPACT. In *European Society for Engineering Education (SEFI) 37th Annual Conference*, 2009.

26. H. Danielsson, editor. *IMPACT — Strategic Development of Chalmers Master's Programmes*, chapter Learning from IMPACT & Quality Assurance, pages 23–24, 59–62. Chalmers, 2010. ISBN 978-91-633-6202-6. Patrik Jansson wrote the chapters *Learning from IMPACT* and *Quality Assurance*. Available from `http://publications.lib.chalmers.se/cpl/record/index.xsql?pubid=115021`.

27. C. Jaeger, P. Jansson, S. van der Leeuw, M. Resch, J. D. Tabara, and R. Dum. GSS orientation paper – background material. Prepared as part of the effort to define a research area within the upcoming EU framework programme Horizon 2020., June 2013.

28. C. Jaeger, P. Jansson, S. van der Leeuw, M. Resch, and J. D. Tabara. GSS: Towards a research program for Global Systems Science. Available from `http://blog.global-systems-science.eu/?p=1512`., 2013. ISBN 978.3.94.1663-12-1. Conference Version, prepared for the Second Open Global Systems Science Conference June 10-12, 2013, Brussels.

4. Patents, state date of registration.

None.

5. Open access computer programs that you have developed

I designed and implemented a compiler for the polytypic language PolyP (which I was later involved in retiring in favour of its successor; Generic Haskell):

29. P. Jansson and U. Norell. The PolyP 2 compiler. Available from `http://www.cse.chalmers.se/~patrikj/poly/`, 2004.

I have participated in the development of the Agda proof engine (mainly through my PhD students Ulf Norell, Nils Anders Danielsson and Jean-Philippe Bernardy). The first description of Agda was in the PhD thesis of my student Ulf Norell (2007) and it has been cited $\simeq 50$ times / year since then, indicating a quick spread in academia.

30. U. Norell et al. Agda — a dependently typed programming language. Implementation available from `http://wiki.portal.chalmers.se/agda/`, 2008.

I have developed a library for specifying and checking algebraic laws of Haskell type classes.

31. P. Jansson and J. Jeuring. The haskell package *ClassLaws*. `http://hackage.haskell.org/package/ClassLaws`, 2012.

I have also contributed to several other libraries and tools:

- 2004–: ChasingBottoms: A library for working with partial and infinite values in Haskell: `http://hackage.haskell.org/package/ChasingBottoms`
- 2006: The BNF Converter — a tool for generating a compiler skeleton from a labelled BNF grammar: `http://bnfc.digitalgrammars.com/`
- 2012: testing-feat: a library for efficiently enumerating large abstract syntax tree types: `http://hackage.haskell.org/package/testing-feat`

6. Popular science articles/presentations

32. P. Jansson and T. Fülöp. A sustainable energy future through education and research. Presented at the G20 Youth Forum Conference held in St.Petersburg, Russia, 2013-04-17/21. Available from `http://wiki.portal.chalmers.se/cse/pmwiki.php/FP/SustainableEnergyFuture`, 2013.

Invited presentations (2006–2013)

2006-12-12: "Testing properties of generic functions" at the Working Group 2.1 on Algorithmic Languages and Calculi, meeting #62 in Namur, Belgium.

2007-09-12: "Comparing Libraries for Generic Programming in Haskell" at the Working Group 2.1 on Algorithmic Languages and Calculi, meeting #63 in Kyoto, Japan.

2007-09-12: "Agda tutorial" at the Working Group 2.1 on Algorithmic Languages and Calculi, meeting #63 in Kyoto, Japan.

2009-07-02 : "Pedagogical development of Master's Programmes for the Bologna Structure at Chalmers - IMPACT" at the 2009 Annual conference of the European Society for Engineering Education (SEFI), Rotterdam, the Netherlands.

2010-01-25: "Parametricity and Dependent Types" at the Working Group 2.1 on Algorithmic Languages and Calculi, meeting #65 in Braga, Portugal.

2010-09-20: "Simple Pure Type System Examples" at the Working Group 2.1 on Algorithmic Languages and Calculi, meeting #66 in Atlantic City, New Jersey, USA.

2011-09-22: "Embedded Parser Generators" at the 2011 ACM SIGPLAN Haskell Symposium, Tokyo, Japan.

2012-10-09: "Functional Enumeration of Algebraic Types" at the IFIP Working Group 2.1 on Algorithmic Languages and Calculi, meeting #69 in Ottawa, Canada.

2012-11-09: "Computer Science meets Global Systems Science" at the 1st Open Global Systems Science Conference, November 8th–10th, 2012, Brussels, Belgium.

2013-06-10: Plenary presentation on "ICT for Global Systems Science" at the 2nd Global Systems Science Conference in Brussels, Belgium.

2013-06-11: "ICT for Global Systems Science" at the Global Systems Science Languages workshop (part of the 2nd Global Systems Science Conference) in Brussels, Belgium.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2013-2993-109043-9

Dnr
2013 -

Name of applicant
Jansson, Patrik

Date of birth
720311-7515

Reg date
2013-09-10 19:39:55

Project title
Strongly Typed Libraries for Programs and Proofs

NT

Grants for distinguished Professors 2013
Grants for distinguished professors
* Övrigt generellt
Chalmers tekniska högskola

NT-RP

2014 2015 2016 2017 2018

2014-01-01 -- 2023-12-31

Appendix S

Signatures

Applicant | **Date**

Head of department at host University | **Clarifi cation of signature** | **Telephone**

A signature on the application is required not only from the applicant but also from the authorised representative of the administrating organisation/institution or equivalent (normally the head of the department or establishment where the research is to be conducted). The signature confirms that the department can accommodate the proposed research, position or equipment; that the costing in the application is approved for the department's part, that any proposed experimentation on human or animal subjects has been reported, and that the applicant has reported any secondary occupations and commercial ties (s)he may have, and nothing inconsistent with good research practice has thereby emerged. The applicant must have discussed these conditions with the representative of the administrating organisation/institution or equivalent before the latter approves and signs the application.

NOTE the exceptions!
Please note that a different procedure applies for the following types of grants:

- Infrastructure- Operation Grants
- Infrastructure- Grants for Expensive Equipment
- Infrastructure- Grants for Large Databases
- Framework Grant SIMSAM
- International Recruitment

For these types of grants the vice-chancellor, instead of the head of department, must sign Appendix S. Thereby (s)he confirms the commitments of the administrative organization. The details are specified in the Call for Proposals, published at www.vr.se.

Vetenskapsrådets noteringar
Kod

2013-2993-109043-9

Vetenskapsrådet, Box 1035, SE-101 38 Stockholm, tel. +46 (0)8 546 44 000, vetenskapsradet@vr.se