VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kansliets noteringar Kod	Dnr
2013-2993-105127-34	

2013
Project Research Grant

Area of science
Natural and Engineering Sciences

Announced grants
Research grants NT April 11, 2013

Total amount for which applied (kSEK)

2014	2015	2016	2017	2018
993	1005	1049	1093	1149

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)

WORKING ADDRESS

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

41296 Göteborg, Sweden

ADMINISTRATING ORGANISATION

Administrating Organisation
Chalmers tekniska högskola

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.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2013-2993-105127-34

Name of Applicant
Jansson, Patrik

Date of birth
720311-7515

Abstract language
English

Keywords
Software Technology, Functional Programming, Dependent Types, Program Verification, Generic Programming

Research areas
*Nat-Tek generellt

Review panel
NT-2

Classification codes (SCB) in order of priority
10201, 10205, 10103

Aspects

Continuation grant
Application concerns: Continuation grant
Registration Number: 2011-6164

Application is also submitted to

similar to:

identical to:

ANIMAL STUDIES

Animal studies
No animal experiments

OTHER CO-WORKER

Name(Last name, First name)
Bernardy, Jean-Philippe

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

Date of birth
781215-0790

Gender
Male

Academic title
PhD

Doctoral degree awarded (yyyy-mm-dd)
2011-06-07

Name(Last name, First name)
,

University/corresponding, Department, Section/Unit, Addressetc.

Date of birth

Gender

Academic title

Doctoral degree awarded (yyyy-mm-dd)

Name(Last name, First name)
,

University/corresponding, Department, Section/Unit, Addressetc.

Date of birth

Gender

Academic title

Doctoral degree awarded (yyyy-mm-dd)

Name(Last name, First name)
,

University/corresponding, Department, Section/Unit, Addressetc.

Date of birth

Gender

Academic title

Doctoral degree awarded (yyyy-mm-dd)

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2013-2993-105127-34

Name of Applicant
Jansson, Patrik

Date of birth
720311-7515

ENCLOSED APPENDICES

A, B, C, D, N, S

APPLIED FUNDING: THIS APPLICATION

Funding period (planned start and end date)
2014-01-01 -- 2017-12-31

Staff/ salaries (kSEK)

Main applicant	**% of full time in the project**	**2014**	**2015**	**2016**	**2017**	**2018**
Patrik Jansson	20	286	296	307	318	329
Other staff						
New PhD student	80	554	573	594	615	637
Total, salaries (kSEK):		840	869	901	933	966

Other projectrelated costs (kSek)	**2014**	**2015**	**2016**	**2017**	**2018**
Travel costs	70	70	70	70	70
Computer equipment	20			20	
PhD examination			10		40
Premises	54	56	58	60	62
Direct IT costs	9	10	10	10	11
Total, other costs (kSEK):	153	136	148	160	183

Total amount for which applied (kSEK)

2014	2015	2016	2017	2018
993	1005	1049	1093	1149

ALL FUNDING

Other VR-projects (granted and applied) by the applicant and co-workers, if applic. (kSEK)

Proj.no.(M) or reg.nr.	**Funded 2013**	**Funded 2014**	**Applied 2014**
2011-6164	800		
Project title	**Applicant**		
Strongly Typed Libraries for Programs and Proofs	Chalmers		

Funds received by the applicant from other funding sources, incl ALF-grant (kSEK)

Funding source	**Total**	**Proj.period**	**Applied 2014**
SSF	25000	2011-2016	
Project title	**Applicant**		
RAW FP: Productivity and Performance through Resource Aware Functional Programming	Chalmers		

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2013-2993-105127-34

Name of Applicant
Jansson, Patrik

Date of birth
720311-7515

POPULAR SCIENCE DESCRIPTION

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
2013-2993-105127-34

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

Patrik Jansson and Jean-Philippe Bernardy

1 Main objectives

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: simple divide-and-conquer algorithms, optimisation problems (inspired by the Algebra of Programming [Bird and de Moor, 1997]), and testing.

2 Research area overview

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 is directly correlated 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 such as Java, C++, Scheme and Haskell are actively gaining abstraction power with Java Generics [Bracha et al., 1998], C++ Templates [Stepanov and Lee, 1995], Scheme's composable macros [Flatt, 2002] and Haskell meta-programming [Sheard and Peyton Jones, 2002]. But there is a danger lurking—more complex features can increase the risk of bugs and unintended behaviour. 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 track of different kinds of values and to keep software from going wrong. In a nutshell, types enable the programmer to keep track of the structure of data and computation in a way that is checkable by the computer itself. Effectively, they act as contracts between the implementor of a program part and its users. If type-checking is performed statically, when the program is compiled, it then amounts to proving that properties hold for all executions of the program, independently of its input.

By the Curry–Howard correspondence, type systems are isomorphic to logics. Rich type systems, such as those 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 [The Coq development team, 2010].

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 [Leroy, 2009]. Other applications are however rapidly appearing. Chlipala et al. [2009] show how to develop and verify imperative programs within Coq. Oury and Swierstra [2008] describe a library for database access which statically guarantees that queries are consistent with the schema of the underlying database. Swamy et al. [2011] show how to implement distributed programming with dependent types. Brady and Hammond [2012] use dependent types to implement resource-safe programs (in the language Idris).

Agda. The programming language Agda is a system based on Martin-Löf type-theory [Martin-Löf, 1984]. Within it, one can express programs, functional specifications as types, and proofs (for example using algebraic reasoning) in a single language (by taking advantage of the Curry–Howard correspondence). Agda is currently emerging as a lingua-franca of programming with dependent types. Its canonical reference, Norell's Thesis [Norell, 2007], has been cited 50 times per year since its publication indicating strong academic interest. 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. Strongly typed languages, such as 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 (sometimes complemented by extensions such as Ssreflect [Gonthier, 2009]). 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 [Chlipala et al., 2009, Leroy, 2009] 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 [Danielsson,

2010], Algebra of Programming [Mu et al., 2009] and Cryptography (ongoing work by N. Pouillard in the `DemTech.dk` project).

In the current Agda implementation, the portions of the library dedicated to programming are essentially decoupled from the portions dedicated to proofs. 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.

3 Project description

Our project will be organised in multiple iterations, each refining the libraries obtained during the previous one. (The first iteration will be based on our current experience with libraries for Haskell and Agda.) 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 application domain (see below for the chosen case studies on algebra of parallel programming, optimisation and testing).

2. **Extraction of common patterns into libraries.** In this phase, we will identify common patterns found in the programs and specifications produced in the previous phase, and capture them in libraries. At the same time, we will tie each pattern of specification to a pattern of implementation. We will then reimplement the application previously produced using the software components of the library.

3. **Refinement of the programming 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.

We work iteratively towards the following milestones (case studies) ranging from classical problems of computer science to domain-specific applications:

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

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

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 strings to just a natural number (the number of white-space separated words in the string). Addition on naturals is associative but a single natural number does not provide enough information to construct a monoid homomorphism. We need to keep the number of full words, plus some information about spacing on either side.

$$
\begin{array}{lcl}
\mathit{helper} & : & \mathit{String} \to \mathit{CountAndSpacing} \\
\mathit{countSpaces} & : & \mathit{CountAndSpacing} \to \mathbb{N} \\
\ldots & & \\
\mathit{wordCount} & : & \mathit{String} \to \mathbb{N} \\
\mathit{wordCount} & = & \mathit{countSpaces} \circ \mathit{helper}
\end{array}
$$

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 show that this works even for something as sequential-looking as parsing (a paper on "Efficient Parallel and Incremental Parsing of Practical Context-Free Languages" is in submission to ICFP 2013).

Domain specific modelling (DSM): 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 [Lincke et al., 2009], grammars for language processing [Duregård and Jansson, 2011], and Walras equilibria and Pareto-efficiency for economics [Ionescu and Jansson, 2013a]. In addition to domain knowledge (provided by our contacts in Potsdam), this requires specifications and proofs for higher-order constructions like monads, functors and natural transformations. In this project we will develop specifications and libraries for 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. An inspiration here could be the syntactic support for domain specific languages in Idris [Brady and Hammond, 2012].

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 will aim to give a more 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 [Duregård et al., 2012] we presented a theory 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 (for example QuickCheck generators) and exhaustive enumeration (in the style of SmallCheck). In this project we want to port this library and its specification to Agda and extend it towards inductive families. Our hypothesis is that, compared the QuickCheck, the more algebraic enumeration approach will be easier to specify and prove correct. (When successful, we may also extend the proofs to QuickCheck.)

4 Preliminary findings

We have published results showing relevant related experience in all the suggested iteration phases and application areas as indicated below.

4.1 The three phases of the iteration

Proven-correct applications: We have worked on correct applications in Haskell [Danielsson and Jansson, 2004, Jansson and Jeuring, 2002] and supporting theory [Danielsson et al., 2006]. We have also worked on applications to climate impact research and economic modelling directly in Agda: [Ionescu and Jansson, 2013a,b].

Patterns into libraries: We have developed, implemented and compared libraries of generic functions [Jansson and Jeuring, 1998a,b, Norell and Jansson, 2004, Rodriguez et al., 2008]. 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 [Bernardy, 2009, Duregård and Jansson, 2011], testing [Duregård et al., 2012, Jeuring et al., 2012] and the above mentioned applications to climate and economy.

Refinement of programming languages: We have designed a generic programming language extension (PolyP [Jansson and Jeuring, 1997]) for Haskell, and we have been involved in the design of the Agda language [Norell, 2007]. We are active in the development of Agda: from the development of parametricity theory [Bernardy and Moulin, 2012, Bernardy et al., 2012], a new kind of generic programming, based on a generalisation of erasure, is being developed. A description and analysis of a core language exemplifying this idea is in submission to ICFP 2013 [Bernardy and Moulin, 2013].

We have also contributed to the development of the "Concepts" feature of C++ by an extensive comparison to Haskell's type classes [Bernardy et al., 2010c].

4.2 The three application areas

AoPP: We have worked actively on implementing programs and proofs in the Algebra of Programming tradition [Backhouse et al., 1999, Mu et al., 2008]. Recent work includes an efficient sparse matrix based matrix algorithm for parallel parsing and its proof (in submission to ICFP 2013).

DSM: 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 and de Moor [1997] 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 [Mu et al., 2009] 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. We also have very recent experience in domain modelling in Agda [Ionescu and Jansson, 2013a,b]. In this project we want to go further in this direction and develop useful libraries of programs and proofs with corresponding types and theorems.

Test: We have explored the tension between testing and proving of higher-order properties [Ionescu and Jansson, 2013b, Jansson et al., 2007], developed a technique for drastically reducing the number of tests required for polymorphic properties [Bernardy et al., 2010a], developed a library for specifying and testing class laws [Jeuring et al., 2012] and a library for functional enumeration [Duregård et al., 2012].

4.3 Other relevant experience

Parametricity theory and applications. Thanks to the Curry-Howard correspondence, the type of each program correspond to a theorem. There is another relationship between types and propositions: each type-assignment gives rise to another theorem (the parametricity condition) about the object being typed. Bernardy and Lasson [2011], Bernardy and Moulin [2012], Bernardy et al. [2010b, 2012] have investigated how to integrate the above result in dependently typed languages. In that context, the net effect is that for every type given by the programmer, an additional property becomes available (for free) for showing the correctness of the program. An interesting application of parametricity is in property based testing of polymorphic functions [Bernardy et al., 2010a]. By combining the results from this paper with our later results on parametricity for dependent types [Bernardy et al., 2012] we want to develop 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 leveraging the potential of 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. A further advantage of this approach is that the skills required to construct programs are directly applicable to understanding the specifications.

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 software prototypes (the libraries and other associated code will be available under an open licence), conference and journal papers and talks (on the techniques used to create the libraries as well as on the amendments made to the languages with dependent types), and doctoral training. We also hope to help the wider research community by contributing libraries for increasingly correct scientific computing.

6 International and national collaboration

With this project, we believe we are in an ideal situation for collaboration, as we have contacts both upstream with the implementors of dependently-typed languages, and downstream with end-users of frameworks for formal modeling and implementation. In fact, we believe that we are in the position to fill in the niche of producing libraries for dependently-typed languages, which are in demand from both sides, but currently lacking.

On the upstream side, we are in direct contact with the group currently in charge of the development of Agda: The main developers, Norell and Danielsson, were Jansson's students; and Agda Implementors' Meetings are held yearly at Chalmers. These meetings regularly attract participants from research groups in Nottingham Univ., Copenhagen ITU, TU Munich, and AIST (in Japan), among others. We have also close contacts with the programming-logic group at Univ. of Gothenburg, which deals with the fundamental aspects of type-theory. (There is a VR-funded multi-project (2013–2016) lead by T. Coquand which will work on Univalent foundations of mathematics, Game theory and Agda.)

Downstream, we have contacts with domain experts at the Potsdam Institute for Climate Impact Research (PIK), which are in demand of tools to describe models of various dynamical systems (such as the atmosphere or the economy) in formal ways, as well as efficient

implementations of these models. Since political decisions may depend on the outcome of their simulations, matching the implementation with the models is important.

7 Organisation and budget

The project is led by Patrik Jansson in the Functional Programming (FP) group of the Computer Science and Engineering (CSE) department at Chalmers. The work will be carried out by Jansson (20%), J-P Bernardy (Ass. Prof. not paid by the project), a PhD student (80%) and several MSc thesis students (not paid by the project). We apply for 70% of the total project cost from VR, the rest is covered by Chalmers and other sources. We will benefit from work on high-level modelling and scientific computing done at (and funded by) PIK (Cezar Ionescu, Nicola Botta).

The first year of project is devoted to library support for Algebra of Programming (milestone **AoPP**), the second and third year focus is on **DSM** and the last years **Test**. Jansson and Bernardy will together supervise the PhD student towards her PhD on "Strongly Typed Libraries for Programs and Proofs".

Jansson and Bernardy are partially funded (20% and 25%) by J. Hughes' "RAWFP: Resource aware functional programming" (SSF, 2011–2016). Hughes' project applies functional programming techniques, especially DSLs embedded in Haskell, to the design and verification of complex software, taking motivating examples from the telecom and automotive domains. The current project proposal, on the other hand, will provide more long-term basic research in the software technology of the future.

References

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.

J.-P. Bernardy. Lazy functional incremental parsing. In *Proc. of the 2nd ACM SIGPLAN symposium on Haskell*, pages 49–60. ACM, 2009.

J.-P. Bernardy and M. Lasson. Realizability and parametricity in pure type systems. In M. Hofmann, editor, *FoSSaCS*, volume 6604 of *LNCS*, pages 108–122. Springer, 2011.

J.-P. Bernardy and G. Moulin. A computational interpretation of parametricity. In *Proc. of the Symposium on Logic in Comp. Sci.* IEEE, 2012.

J.-P. Bernardy and G. Moulin. Type-theory in color, 2013. submitted to ICFP 2013.

J.-P. Bernardy, P. Jansson, and K. Claessen. Testing polymorphic properties. In A. Gordon, editor, *European Symposium on Programming*, volume 6012 of *LNCS*, pages 125–144. Springer, 2010a.

J.-P. Bernardy, P. Jansson, and R. Paterson. Parametricity and dependent types. In *Proc. of ICFP 2010*, pages 345–356. ACM, 2010b.

J.-P. Bernardy, P. Jansson, M. Zalewski, and S. Schupp. Generic programming with C++ concepts and Haskell type classes—a comparison. *J. Funct. Program.*, 20(3–4):271–302, 2010c. doi: 10.1017/S095679681000016X.

J.-P. Bernardy, P. Jansson, and R. Paterson. Proofs for free — parametricity for dependent types. *J. Funct. Program.*, 22(02):107–152, 2012.

R. Bird and O. de Moor. *Algebra of Programming*, volume 100 of *International Series in Computer Science*. Prentice-Hall International, 1997.

G. Bracha, M. Odersky, D. Stoutamire, and P. Wadler. Making the future safe for the past: adding genericity to the java programming language. In *OOPSLA '98*, pages 183–200. ACM, 1998. doi: 10.1145/286936.286957.

E. Brady and K. Hammond. Resource-safe systems programming with embedded domain specific languages. In *Practical Aspects of Declarative Languages*, pages 242–257. Springer, 2012.

A. Chlipala, G. Malecha, G. Morrisett, A. Shinnar, and R. Wisnesky. Effective interactive proofs for higher-order imperative programs. In *Proc. of ICFP 2009*, ICFP '09, pages 79–90. ACM, 2009.

N. A. Danielsson. Total parser combinators. In *Proc. of ICFP 2010*, ICFP '10, pages 285–296. ACM, 2010.

N. A. Danielsson and P. Jansson. Chasing bottoms, a case study in program verification in the presence of partial and infinite values. In *MPC 2004*, volume 3125 of *LNCS*, pages 85–109. Springer, July 2004.

N. A. Danielsson, J. Hughes, P. Jansson, and J. Gibbons. Fast and loose reasoning is morally correct. In *POPL'06*, pages 206–217. ACM Press, 2006.

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.

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.

M. Flatt. Composable and compilable macros:: you want it when? In *ICFP '02*, pages 72–83. ACM, 2002. doi: 10.1145/581478.581486.

G. Gonthier. Ssreflect: Structured scripting for higher-order theorem proving. In *PLMMS'09*, page 1. ACM, 2009.

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, 2013a.

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, 2013b. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi: 10.4230/LIPIcs.TYPES.2011.41.

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.

P. Jansson and J. Jeuring. PolyLib – a polytypic function library. Workshop on Generic Programming, Marstrand, 1998a. Available from `www.cse.chalmers.se/~patrikj/poly/polylib/`.

P. Jansson and J. Jeuring. Functional pearl: Polytypic unification. *J. Funct. Program.*, 8(5): 527–536, 1998b.

P. Jansson and J. Jeuring. Polytypic data conversion programs. *Science of Computer Programming*, 43(1):35–75, 2002.

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.

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

X. Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

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.

P. Martin-Löf. *Intuitionistic type theory.* Bibliopolis, 1984.

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.

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.

U. Norell. *Towards a practical programming language based on dependent type theory.* PhD thesis, Chalmers Tekniska Högskola, 2007.

U. Norell and P. Jansson. Polytypic programming in Haskell. In *Implementation of Functional Languages 2003*, volume 3145 of *LNCS*, pages 168–184. Springer, 2004.

N. Oury and W. Swierstra. The power of Pi. In *Proc. of ICFP 2008*, pages 39–50. ACM, 2008.

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.

T. Sheard and S. Peyton Jones. Template meta-programming for haskell. In *Proc. of the 2002 ACM SIGPLAN workshop on Haskell*, Haskell '02, pages 1–16. ACM, 2002.

A. A. Stepanov and M. Lee. The standard template library. Technical Report HPL-95-11(R.1), Hewlett Packard Laboratories, Palo Alto, CA, USA, Nov. 1995.

N. Swamy, J. Chen, C. Fournet, P.-Y. Strub, K. Bhargavan, and J. Yang. Secure distributed programming with value-dependent types. In *Proceeding of the 16th ACM SIGPLAN international conference on Funct. Programming, ICFP 2011, Tokyo, Japan, September 19-21, 2011*, pages 266–278, 2011.

The Coq development team. The Coq proof assistant, 2010.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2013-2993-105127-34

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

C Curricula Vitæ

Curriculum Vitæ: Patrik Jansson, 1972-03-11

1. Higher education degree:

1995: BSc+MSc degrees in Eng. Physics + Eng. 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 CS from Chalmers, Sweden, on *Functional Polytypic Programming*, Advisor: Johan Jeuring.

3. PostDoc and guest research:

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.

4. Qualification as Assoc. Professor:

2004: Docent (Associate Prof.) degree from Chalmers, Sweden.

5. Current Employment:

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

6. Prev. Employment and Education:

2001–2004: Ass. Prof. in CS, Chalmers.

2004–2011: Associate Professor, Chalmers.

7. Interruptions in research:

Parental leave with Julia (1999) and Erik (2004) for a total of one full time year.

2002–2005: Director of studies of the CS dept. On average 35% / year for three years.

2005–2008: Vice head of the CSE dept. On average 50% of full time / year for four years.

2011–2013: Head of the 5y CSE programme. On average 42% of full time / year.

8. Supervision experience:

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 supervised over 20 MSc and BSc project students.

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 defenses at Chalmers (T. Gedell, CSE (2008), M. Zalewski, CSE (2008), H. Johansson, Physics (2010)).

9. Awards, grants, etc.:

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

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

1998: Organiser of the first Workshop on Generic Programming (WGP), Marstrand.

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.

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 WGP.

2009: PC Chair for WGP

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 ICT).

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 FET Open).

2011–2016: Co-applicant on "RAW FP: Productivity and Performance through Resource Aware Functional Programming" (RAW FP) 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.

2011: Organiser of a workshop on "Domain Specific Languages for Economical and Environmental Modelling (DSL4EE)" in Marstrand as part of GSDP.

2011–2012: Steering Committee Chair of WGP.

2012: Workshops chair of the International Conference on Functional Programming (ICFP 2012).

2012: Organised two "ICT challenges to Global Systems Science" workshops in Brussels as part of the First Open Global Systems Science Conference.

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 ICFP 2013.

I have been reviewer for Journal of Functional Programming, Science of Computer Programming, Principles of Programming Languages, ICFP, IFL and several other journals and conferences.

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 CS department

2005–2008: Vice head of the CSE 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 GU.

Curriculum Vitæ: Jean-Philippe Bernardy, 19781215-0790

1. Higher education degree:

1996–2000: BSc+MSc degrees in CS, obtained with "la plus grande distinction" (highest distinction), Université Libre de Bruxelles, July 2000.

2. Doctoral degree:

2011: Ph.D. degree in CS from Chalmers, Sweden. Thesis: *A Theory of Parametric Polymorphism and an Application*, Advisor: Patrik Jansson.

3. PostDoc:

2011–2012: Continued development of the theory of parametric polymorphism, in collaboration with Prof. Thierry Coquand and Prof. Peter Dybjer (Chalmers).

4. Docent degree:

I plan to obtain my Docent degree before 2015.

5. Current Employment:

2012–now: Assistant Prof., Chalmers. Research 75% (2013).

6. Prev. Employment:

2007–2011: Doctoral Student, Chalmers, Sweden
2005–2007: Software Engineer, Eurocontrol (Brussels)
2000–2003: Software Engineer, PhiDaNi Software (Brussels)

7. Interruptions in research:

I have taken 56 days of parental leave during my employment as Assistant Professor.

8. Supervised PhD and PostDoc:

I am currently co-supervising Guilhem Moulin. The main supervisor is Peter Dybjer.

9. Awards, grants, etc.:

Co-applicant on *Types for programs and proofs* funded with **12 M SEK** by the Swedish Science Council.

Invited talks:

- "Unobtrusive Version Control", Potsdam Institute for Climate Impact Research, 2007
- "Concepts and Type-Classes", Workshop on Generic Programming, 2008
- "Yi: the Haskell editor", Haskell Symposium, 2008
- "Testing Polymorphic Properties" European Symposium on Programming, 2009
- "Parametricity and Dependent types", International Conference of Functional Programming, 2010
- "Proof-Irrelevance in Agda", Agda Interest Meeting, Nottingham 2010
- "Realisability and Parametricity in PTSs", Microsoft Research, Cambridge, 2010

- "Internalizing Parametricity", Agda Interest Meeting, Shonan Village, Japan, 2011
- "Implementing Parametricity", Parametricity Workshop, Glasgow 2012
- "Type-Theory in Color", Agda Interest Meeting, Copenhagen 2012

Teaching Experience:

I have been responsible for teaching the following courses:

- Programming Paradigms (2012, 2013).
- Functional Programming Languages with Linear Types (2013).

Implementations:

I am the main developer of the Yi editor.
I have made significant contributions to the following industrial-strength tools:

- Alex (Lexer generator)
- BNFC (Parser generator)
- Agda

Community roles:

- Haskell Symposium 2013, PC Member
- Haskell Implementers Workshop 2010, PC Member

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2013-2993-105127-34

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

C Publication lists

The publications most relevant for this project are marked with an arrow (⇒).

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.

Most cited publications (Google Scholar, 2013-03-25)

Jansson's Hirsch-index is **16**, his total citation count is over **1200** and the following papers are the five most cited (not including the papers in the last 8 years).

- 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.
  Number of citations: **307**.

- 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.
  Number of citations: **182**.

- 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.
  Number of citations: **158**.

- 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.
  Number of citations: **56**.

⇒ P. Jansson and J. Jeuring. Polytypic data conversion programs. *Science of Computer Programming*, 43(1):35–75, 2002.
Number of citations: **50**.

Journal articles (last 8 years, excluding the above)

- J.-P. Bernardy, P. Jansson, and R. Paterson. Proofs for free — parametricity for dependent types. *J. Funct. Program.*, 22(02):107–152, 2012.
  Number of citations: **4**.

- J.-P. Bernardy, P. Jansson, M. Zalewski, and S. Schupp. Generic programming with C++ concepts and Haskell type classes — a comparison. *J. Funct. Program.*, 20 (3–4):271–302, 2010c. URL `http://dx.doi.org/10.1017/S095679681000016X`.
  Number of citations: **10**.

⇒ 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: **7**.

Articles in refereed collections and conf. proceedings (last 8 years)

⇒ 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, 2013a.
Number of citations: **0**.

- 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, 2013b. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi: 10.4230/LIPIcs.TYPES.2011.41.
  Number of citations: **0**.

- 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: **4**.

- 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**.

- 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: **3**.

⇒ J.-P. Bernardy, P. Jansson, and R. Paterson. Parametricity and dependent types. In *Proc. of ICFP 2010*, pages 345–356. ACM, 2010b.
Number of citations: **25**.

⇒ J.-P. Bernardy, P. Jansson, and K. Claessen. Testing polymorphic properties. In A. Gordon, editor, *European Symposium on Programming*, volume 6012 of *LNCS*, pages 125–144. Springer, 2010a.
Number of citations: **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.
Number of citations: **56**.

- 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.
Number of citations: **5**.

- 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, 2008a.
Number of citations: **20**.

- 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.
Number of citations: **12**.

- 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.
Number of citations: **4**.

- N. A. Danielsson, J. Hughes, P. Jansson, and J. Gibbons. Fast and loose reasoning is morally correct. In *POPL'06*, pages 206–217. ACM Press, 2006.
Number of citations: **51**.

Publicly available implementations (last 8 years)

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),

- U. Norell et al. Agda — a dependently typed programming language. Implementation available from Google Code: `http://code.google.com/p/agda/`, 2008.

  The first description of Agda was in the PhD thesis of Ulf Norell (2007) and it has been cited $\simeq 50$ times / year since then, indicating a quick spread in academia.

Publication list for Jean-Philippe Bernardy

Database used for citation data: Google scholar.

Peer-reviewed publications in journal

⇒ J.-P. Bernardy, P. Jansson, and R. Paterson. Proofs for free — parametricity for dependent types. *J. Funct. Program.*, 22(02):107–152, 2012.
Number of citations: **4**.

- J.-P. Bernardy, P. Jansson, M. Zalewski, and S. Schupp. Generic programming with C++ concepts and Haskell type classes — a comparison. *J. Funct. Program.*, 20 (3–4):271–302, 2010c. URL `http://dx.doi.org/10.1017/S095679681000016X`.
Number of citations: **10**.

Peer-reviewed publications in conferences and workshops

⇒ J.-P. Bernardy and G. Moulin. A computational interpretation of parametricity. In *Proc. of the Symposium on Logic in Comp. Sci.* IEEE, 2012.
Number of citations: **4**.

- J.-P. Bernardy and M. Lasson. Realizability and parametricity in pure type systems. In M. Hofmann, editor, *FoSSaCS*, volume 6604 of *LNCS*, pages 108–122. Springer, 2011.
Number of citations: **11**.

⇒ J.-P. Bernardy, P. Jansson, and R. Paterson. Parametricity and dependent types. In *Proc. of ICFP 2010*, pages 345–356. ACM, 2010b.
Number of citations: **25**.

⇒ J.-P. Bernardy, P. Jansson, and K. Claessen. Testing polymorphic properties. In A. Gordon, editor, *European Symposium on Programming*, volume 6012 of *LNCS*, pages 125–144. Springer, 2010a.
Number of citations: **16**.

⇒ J.-P. Bernardy. Lazy functional incremental parsing. In *Proc. of the 2nd ACM SIGPLAN symposium on Haskell*, pages 49–60. ACM, 2009.
Number of citations: **8**.

- J.-P. Bernardy, P. Jansson, M. Zalewski, S. Schupp, and A. Priesnitz. A comparison of C++ concepts and Haskell type classes. In *WGP '08: Proc. of the ACM SIGPLAN workshop on Generic programming*, pages 37–48. ACM, 2008b.
Number of citations: **20**.

Non peer-reviewed publications

- J.-P. Bernardy. Yi: an editor in Haskell for Haskell. In *Proc. of the first ACM SIGPLAN symposium on Haskell*, pages 61–62. ACM, 2008.
  Number of citations: **6**.

Publicly available implementations

I am the main contributor to the Yi project

- J.-P. Bernardy. Yi: an editor in Haskell for Haskell. In *Proc. of the first ACM SIGPLAN symposium on Haskell*, pages 61–62. ACM, 2008.

I have contributed the Agda proof engine. The first description of Agda was in the PhD thesis of Ulf Norell and it has been cited $\simeq 50$ times / year since then, indicating a quick spread in academia.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2013-2993-105127-34

Name of applicant
Jansson, Patrik

Date of birth
720311-7515

Title of research programme

Appendix D

Scientific report/account of scientific activity

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

D Scientific Report

The main focus of the research project was on *libraries* to help develop programs with matching specifications. The results are in three tracks: Haskell libraries, Agda libraries and more fundamental results in parametricity theory. We have published three Haskell Symposium papers, each contributing one such library and two papers about Agda libraries. In parallell with the more applied case studies we also worked on more fundamental theoretical results which extend parametricity theory to apply to dependent types (constructive logic). This resulted in two published papers and a prototype implementation in Agda.

Embedded Parser Generators The paper describes how to embed context-free grammars in Haskell so that parsers and pretty-printers are automatically generated from them. In this way we combine features of parser generators (static grammar checks, clear BNF syntax) and features that are otherwise exclusive to combinator libraries (reuse, parameters, grammar generation inside Haskell). The library (BNFC-meta) enables domain experts to focus on describing the grammar (a specification of the intended language) in a variant of BNF without worrying about the implementation details.

Testing Type Class Laws The specification of a class in Haskell often starts with stating, in comments, the laws that should be satisfied by methods defined in instances of the class, followed by the type of the methods of the class. This paper develops a library that supports testing such class laws using QuickCheck. Our library is a light-weight class law testing framework, which requires a limited amount of work per class law, and per datatype for which the class law is tested.

Feat: Functional Enumeration of Algebraic Types In mathematics, an enumeration of a set S is a bijective function from (an initial segment of) the natural numbers to S. We define "functional enumerations" as efficiently computable such bijections. This paper describes a theory of functional enumeration and provides an algebra of enumerations closed under sums, products, guarded recursion and bijections. We implement our ideas in a Haskell library called testing-feat, and make the source code freely available. Feat provides efficient "random access" to enumerated values. The primary application is property-based testing, where it is used to define both random sampling (for example QuickCheck generators) and exhaustive enumeration (in the style of SmallCheck).

Testing versus proving in climate impact research Higher-order properties arise naturally in some areas of climate impact research. For example, "vulnerability measures" must fulfill certain conditions which are best expressed by quantification over all increasing functions. This kind of property is notoriously difficult to test. However, for the measures used in practice, it is quite easy to encode the property as a dependent type and prove it correct. Moreover, in scientific programming, one is often interested in correctness "up to implication": the program would work as expected, say, if one would use real numbers

instead of floating-point values. Such counterfactuals are impossible to test, but again, they can be easily encoded as types and proven. We show examples of such situations (encoded in Agda), encountered in actual vulnerability assessments.

Dependently-typed programming in scientific computing Computer simulations are essential in economics where the ability to make laboratory experiments is limited yet it is important to ensure that the models are implemented correctly. Typically, though, the models are only informally specified. We argue that using dependent types allows us to gradually reduce the gap between the mathematical description and the implementation, and we provide a library for specification of basic concepts from economic modelling.

Proofs for free Reynolds' abstraction theorem shows how a typing judgement in System F can be translated into a relational statement (in second order predicate logic) about inhabitants of the type. We obtain a similar result for pure type systems: for any PTS used as a programming language, there is a PTS that can be used as a logic for parametricity. Types in the source PTS are translated to relations (expressed as types) in the target. Similarly, values of a given type are translated to proofs that the values satisfy the relational interpretation.

A computational interpretation of parametricity In this paper, we show how the above theorem can be internalized. More precisely, we describe an extension of Pure Type Systems with a special parametricity rule (with computational content), and prove fundamental properties such as Church-Rosser's and strong normalization. All instances of the abstraction theorem can be both expressed and proved in the calculus itself.

D.1 The relation between the old and the new project

The new project is a natural continuation of the old project but what we have learnt during the way has changed the focus a bit. Where the old project talks about programs and proofs, the new project explicitly includes also specification (and automatic testing) as intermediate stages. As the new project is not bigger, this will mean less focus on programs and more focus on libraries for specifications and proofs.

D.2 Available research resources

The project was granted 2.4M SEK from VR (P. Jansson, diarienummer 2011-6164) which mainly paid for J.-P. Bernardy (PostDoc, 50%) but also 20% of P. Jansson. Jansson (20%) and J. Duregård (80%) were also partially supported by a VR multi-project grant in ICT, 2009–2012 (J. Hughes, diarienummer 2009-4303). Finally the Potsdam Institute for Climate Impact Research supported Cezar Ionescu who worked part-time in this project.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2013-2993-105127-34

Name of applicant
Jansson, Patrik

Date of birth
720311-7515

Title of research programme

Appendix N

Budget and research resources

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

N Budget

I apply for 80% of a PhD student salary (the other 20% are covered by teaching) and for 20% of my own salary. The amounts include indirect costs from the department and university level. I also apply for direct costs for travel money for the new student.

Cost	2014	2015	2016	2017	2018
Patrik Jansson, 20%:	286	296	307	318	329
New PhD student, 80%:	554	573	594	615	637
Travel, offices, IT:	153	135	148	160	183
Total:	993	1004	1048	1093	1149

Jansson is partially funded (20%) by J. Hughes' "RAWFP: Resource aware functional programming" (SSF, 2011–2016). Hughes' project applies functional programming techniques, especially DSLs embedded in Haskell, to the design and verification of complex software, taking motivating examples from the telecom and automotive domains. The current project proposal, on the other hand, will provide more long-term basic research in the software technology of the future.

In addition to teaching (25%), the co-applicant Bernardy is mainly (50%) funded by the ongoing VR project (P. Jansson, 2011–2013) for which we apply for a continuation. He is partially funded (25%) by RAWFP (described above) and he will probably be partially funded (20%) by a recently granted multi-project-grant (VR 2013–2016, number 2012-5294) lead by T. Coquand. Therefore we do not apply for funding from VR for him in this project. Coquand's project will also complement the current project by strengthening the Agda group at U. of Gothenburg.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2013-2993-105127-34

Dnr
2013 -

Name of applicant
Jansson, Patrik

Date of birth
720311-7515

Reg date
2013-04-11 18:59:52

Project title
Strongly Typed Libraries for Programs and Proofs

NT

Research grants NT April 11, 2013
Project Research Grant
*Nat-Tek generellt
Chalmers tekniska högskola

Appendix S

Signatures

NT-2

2014	2015	2016	2017	2018	
993	1005	1049	1093	1149	2014-01-01 -- 2017-12-31

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

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

2013-2993-105127-34

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