Ansökan

2015-03-31

Jansson, Patrik

Information om sökande

Namn: Patrik Jansson

Dr-examen: 2000-06-09

Födelsedatum: 19720311

Akademisk titel: Annan

Kön: Man

Arbetsgivare: Chalmers tekniska högskola

Medelsförvaltare: Chalmers tekniska högskola

Hemvist: 3725 - Software Technology

Information om ansökan

Utlysningsnamn: Forskningsbidrag Stora utlysningen 2015 (Naturvetenskap och teknikvetenskap)

Bidragsform: Projektbidrag

Sökt inriktning: Fri

Ämnesområde:

Projekttitel (svenska): Program du kan lita på: högnivåspecifikationer och korrekta implementationer via beroende typer

Projektstart: 2016-01-01

Projektslut: 2019-12-31

Sökt beredningsgrupp: NT-2, NT-1, NT-14

Klassificeringskod: 10201. Datavetenskap (datalogi), 10103. Algebra och logik

Nyckelord: Software Technology, Functional Programming, Dependent Types, Program Verification, Generic Programming

Sökta medel

År:	2016	2017	2018	2019
Belopp:	997 000	1 000 000	1 083 000	1 083 000

Medverkande

Namn: Jean-Philippe Bernardy

Dr-examen: 2011-06-07

Födelsedatum: 19781215

Akademisk titel: Docent

Kön: Man

Arbetsgivare: Ingen nuvarande arbetsgivare

Namn: Cezar Ionescu

Dr-examen: 2009-02-09

Födelsedatum: 19681221

Akademisk titel: Doktor

Kön: Man

Arbetsgivare: Chalmers tekniska högskola

Beskrivande information

Projektinformation

Projekttitel (svenska)*

Program du kan lita på: högnivåspecifikationer och korrekta implementationer via beroende typer

Projekttitel (engelska)*

Libraries You Can Trust: High-level Specifications and Correct Implementations via Dependent Types

Abstract (Engelska)*

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 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 stronger guarantees of correctness to a wider user base. To check the applicability of our libraries, our case studies are parallel programming by monoid homomorphisms, domain specific modelling of emissions trading and property based testing tools.

Populär-vetenskaplig beskrivning (svenska)*

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).

Projekttid

Antal år för projektet*

4

Beräknad projekttid*

2016-01-01 - 2019-12-31

Klassificeringar

Välj minst en och upp till tre SCB-koder som motsvarar inriktningen för ditt projekt, i prioritetsordning.

Välj SCB-kod i de tre nivåerna och klicka sedan på det nedre plustecknet för att spara ditt val.

SCB-koder*

1. Naturvetenskap > 102. Data- och informationsvetenskap (Datateknik) > 10201. Datavetenskap (datalogi)

1. Naturvetenskap > 101. Matematik > 10103. Algebra och logik

---

Ange minst tre och upp till fem kortfattade nyckelord som beskriver projektet.

Nyckelord 1*

Software Technology

Nyckelord 2*

Functional Programming

Nyckelord 3*

Dependent Types

Nyckelord 4

Program Verification

Nyckelord 5

Generic Programming

Etiska överväganden

Redovisa tydligt de etiska frågor som projektet (eller motsvarande) aktualiserar och redogör för hur de behandlas i forskningsarbetet. Markera också de specifika aspekter som eventuellt berör din ansökan.

Redogörelse för etiska överväganden*

None.

I projektet ingår hantering av persondata

Nej

I projektet ingår djurförsök

Nej

I projektet ingår humanförsök

Nej

Forskningsplan

Libraries You Can Trust:

High-level Specifications and Correct Implementations via Dependent Types

P. Jansson, J.-P. Bernardy and C. Ionescu

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: the Algebra of Parallel Programming (inspired by the Algebra of Programming [Bird and de Moor, 1997]), Domain-specific Modelling, and Testing.

Note that what we call "software" or "program" in this proposal is not just "*a sequence of instructions, written to perform a specified task with a computer*" [Wikipedia, Computer Program]. In our terminology a "program" is a more high-level description of what a computer should compute (perhaps closer to the meaning of "algorithm") or even, in the case of Domain-specific Modelling, a high-level description of a problem or a solution in a particular application domain (be it financial contracts, partial differential equations or formal proofs). This means that the iterative process of coming up with a "program" matching a "specification" is not limited to traditional programming, but it is also a way of exploring and understanding a domain. A resulting library of specifications can be seen as a collection of building-blocks for describing (problems in) the domain and a library of implementations can be seen as computer aided methodologies for computing with, or solving problems within, that domain.

2 Project description

Our project will be organised in multiple iterations, each refining the libraries developed in the previous one. The first iteration is based on our current experience with libraries built in the functional programming languages 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 applications. In this phase, we will assess the viability of our libraries by applying them to different applications ranging from classical problems of computer science to domain-specific applications (details in the following subsections).

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 one or more patterns of implementation. We will then reimplement the application previously produced using the 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.

The iterative refinement steps will be used in three major case studies described in the following subsections: **Algebra of Parallel Programming**, **Domain-specific Modelling**, and **Testing**.

2.1 Algebra of Parallel Programming (AoPP)

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 Bird and de Moor–style program derivation in the dependently typed programming language Agda. A program is coupled with an algebraic derivation from a specification, whose correctness is guaranteed by the type system. We believe that this approach is useful in tackling one of the key problems confronting the computing world today: that of correctly implementing scalable parallel computations. This is our aim in the first case study.

At the core of scalable parallel programming is the ability to divide a workload into two independent tasks (which can be run in parallel) in such a way the solutions can be easily combined into a final result. This subdivision can be done recursively to the depth needed to effectively use the available hardware parallelism. We want the results of the parallel computation to be independent of the number of divisions of the workload, otherwise we would obtain different results on machines with different numbers of processors. Similarly, we want to obtain the same result irrespective of the order in which the individual tasks terminate.

These conditions are met by a large class of algorithms, namely those which are monoid homomorphisms. That is, a function $f : A \to B$ can be parallelised if it satisfies the following laws:

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

where $empty$ and $+\!\!+$ denote monoidal unit and composition (for the type in the subscript).

Often, a function which is not a monoid homomorphism can be formulated in terms of an auxiliary function, which works on an extended type.

A simple example is word counting, which maps strings to natural numbers (the number of white-space separated words in the string). The monoid structures are concatenation with the empty string as unit for the domain, and addition with zero as unit for the codomain. It is easy to see that word counting is not a homomorphism: if we cut a string containing a single word (no spaces) in two, each subtask will count one word, and the addition of the two independent results will return the erroneous count of two.

The reason for that is that we have lost the information about the (lack of) spacing. To avoid this loss, we need to preserve the information about spacing on either side, in addition to counting the number of full words.

$$
\begin{array}{lcl}
helper & : & String \rightarrow CountAndSpacing \\
countSpaces & : & CountAndSpacing \rightarrow \mathbb{N} \\
... & & \\
wordCount & : & String \rightarrow \mathbb{N} \\
wordCount & = & countSpaces \circ helper
\end{array}
$$

In this simple case, inventing the $CountAndSpacing$ type and the $helper$ function does not require much ingenuity. Things change when we move on to more realistic examples, such as that of parallel parsing.

A parser is a program that analyses a piece of text to determine its logical structure. Parsing is, at least at first sight, an inherently sequential process, especially when it comes to formal texts, such as program code, since lines of code often require the surrounding context in order to make sense.

Nevertheless, inspired by sparse matrix algorithms and the work of Valiant [1975] on language recognition, we have been able to create a suitable analogue of the $CountAndSpacing$ type for parsing, leading to "*Efficient Parallel and Incremental Parsing of Practical Context-Free Languages*" [Bernardy and Claessen, 2013].

Perhaps more importantly, when formalising the proofs of correctness of the parallel parsers, we have been able to use the Bird and de Moor approach in order to *calculate* the parallelisation from the specification of parsing[1]. This kind of program calculation is at the core of what we call the Algebra of Parallel Programming (AoPP).

In this project, we aim to develop and use AoPP in order to

- calculate parallel algorithms for optimisation algorithms (and other numerical methods required by the domain-specific modelling case study),
- calculate parallel versions of certain graph algorithms (like path finding),
- develop general principles for the construction of the intermediate datatypes and helper applications.

2.2 Domain-specific modelling (DSM)

What good is proof of correctness 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 more recently, Walras equilibria and Pareto-efficiency for economics [Ionescu and Jansson, 2013a].

In the case of economics, the resulting specifications, while quite close to the mathematical formulations that the modellers are used to, are *non-constructive* in nature and can only be implemented in restricted settings (for example, when all the sets involved are finite). To deal with more general cases will involve a great deal of innovations both in the concepts to be specified (consider the difference between discrete and continuous mathematics) and in the numerical methods to be implemented. To bring about such innovations is our aim in this area of the project.

[1] A paper is under review and the core ideas were presented as "*An algebra for parallel parsing*" at the IFIP Working Group 2.1 on Algorithmic Languages and Calculi, meeting #71 in Zeegse, the Netherlands (March 2014).

The chosen application area for the first phase of our iterative development is (mathematical models of) emissions trading games or, more generally, international environmental agreements. Here we will continue from the work on scientific computing for economics and climate impact by formalizing some of the key concepts (*player, coalition, market, stability, free-riding*). In addition to domain knowledge (provided by our contacts in Potsdam), this requires specifications and proofs for higher-order constructions like monads, functors and vulnerability measures [Ionescu, 2009].

In the second phase this will lead to a library of common patterns which can be seen as a domain specific language for expressing models of emission trading and coalition formation.

And in the third phase we will suggest ways of improving language support to present the specifications in a way accessible to the domain experts. This includes better syntactic support for domain specific languages starting from what is available in the language Idris [Brady and Hammond, 2012], and better support for eliding information starting from the support for hidden arguments in Agda.

2.3 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 to QuickCheck, the more algebraic enumeration approach will be easier to specify and prove correct. (When successful, we may also extend the proofs to QuickCheck.)

The testing tools developed in this case study will also be used in the other case studies in the exploration phase towards finding the right specifications. Experience shows that testing can be very efficient in ruling out bad specification attempts by providing concrete failing test cases.

But note that testing is not only a *tool* used in the project. The domain of testing and enumeration is also a *case study* for the development of libraries you can trust. Following our common iterative process we will 1) develop a proven correct implementation of enumeration of complex AST types, 2) extract the common specification and implementation patterns into a library and 3) refine the underlying language to better support the interplay between testing and proving.

2.4 Organisation

The project is led by Patrik Jansson in the Functional Programming (FP) group of the CSE department at Chalmers. The work will be carried out by Jansson (20%), a PhD student (80%) and by J.-P. Bernardy (AssProf), C. Ionescu (PostDoc), 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) the Potsdam Institute for Climate Impact Research (N. 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 year's focus is **Test**. Jansson and Bernardy will together supervise the PhD student towards her PhD on "High-level Specifications and Correct Implementations via Dependent Types".

To educate MSc and PhD students (in this and in other projects) we plan to organise a summer school in 2016 on the Algebra of Parallel Programming in Agda. We will also build up a github repository of library code (for specifications and implementations) produced in the project.

3 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, C++ Templates, Scheme's composable macros, and Haskell meta-programming. But power always comes at a price: in this case, without proper checking, more complex features can increase the risk of bugs and unintended behaviour. Thus, with new abstraction mechanisms we also need new, preferably computer-aided, mechanisms for checking the program code.

Types. Types are used in many parts of computer science to organise the different kinds of values and to prevent 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 possible executions of the program, independently of its input.

By the Curry–Howard correspondence, type systems are directly related 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 only recently gained popularity as a medium for programming. The viability of dependent types in a substantial "real world" example was perhaps first demonstrated by 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.

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 [2007], has been cited 50 times per year since its publication indicating strong academic interest. Additionally, there are several high-quality video introductions to Agda available on the Internet, produced by scientists with no affiliation to Chalmers. These range from short demonstrations (e.g., L. Maydwell's 15 min. available on YouTube since 2012) to mini-courses consisting of five lectures (e.g., D. Licata's *Dependently-Typed Programming in Agda*, presented at the Oregon Prog. Lang. Summer School in 2013). Accordingly, 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 for programming, 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 `www.DemTech.dk` project).

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]. More recent work (in submission) includes a formalization of Sequential Decision Problems [2] and follow-up work on A computational theory of policy advice and avoidability [3].

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, Bernardy and Claessen, 2013, Duregård and Jansson, 2011], testing [Duregård et al., 2012, Jeuring et al., 2012] and the above mentioned applications to climate and economy.

[2] `http://wiki.portal.chalmers.se/cse/pmwiki.php/FP/SeqDecProb`
[3] `https://github.com/nicolabotta/SeqDecProbs`

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 was shown at ICFP [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., 2010b].

4.2 The three application areas

AoPP: In [Mu et al., 2009] we presented a library for Bird and de Moor–style program derivation in Agda. Based on this work, a similar library has been implemented in Idris by David Christiansen.

We have developed an efficient sparse matrix based algorithm for parallel parsing [Bernardy and Claessen, 2013]. As a follow-up, we have recently shown that it is possible to *calculate* the efficient parallel algorithm from a specification of parsing, thus illustrating the promise of an algebra of parallel programming.

DSM: We have used dependent types in order to express high-level specifications of software components used in computational assessments of vulnerability to climate change [Ionescu, 2014, Ionescu and Jansson, 2013b]. We have been able to prove the correctness of some of these components (and explain why others were incorrect), and, perhaps more importantly, we have been able to clarify some of the terminological confusion existing in the field.

In [Ionescu and Jansson, 2013a], we have also used type theory to specify the basic building blocks of economic theory, used in almost all economic models today, concepts such as Pareto efficiency, Walrasian equilibrium, Nash equilibrium, etc., together with the relations between them (for example, Walrasian equilibria are Pareto efficient). Recently, we have developed specifications and correct-by-construction implementations for a large class of sequential decision problems [Botta et al., 2013].

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].

5 Significance

Effective production of correct software is a problem which remains unsolved, and is of great economic significance. By leveraging the capabilities 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.

Dependently-typed programming languages also allow us to formally encode and verify *program derivations*, which are important in minimising the number of "Eureka" steps needed to go from the specification to the implementation. This is even more so in the context of parallel programming, where the problems of sequential programming are compounded by the need to

invent suitable operations and types for efficient parallelisation. The development of an algebra of parallel programming will be a significant advance in this area.

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, and set new standards for the design of libraries in general.

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.

As the example of parallel parsing shows, there are computations that can be expressed as instances of (abstract) linear algebra algorithms. At the same time, the calculational proofs that we have given for the correctness of these algorithms improve on the classical informal proofs. The connection between computing science and linear algebra has been noticed several times in the literature, but it has until now been exploited mainly in one direction: finding new applications of linear algebra. We aim to explore the connection also in the other direction: using recursive types and calculational proofs to simplify presentations of classical linear algebra notions and results. This will be a natural part of our more general efforts (beyond this project) to specify and implement validated numerical methods.

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 modelling and implementation. In fact, we believe that we are in the position to fill the need for 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: Two of 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 (T. Coquand, U. Norell, N. A. Danielsson, A. Abel).

Downstream, we have contacts with domain experts (N. Botta, J. Heitzig) at the Potsdam Institute for Climate Impact Research (PIK), who need 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, correctly implementing these models is important.

Based on our experience with functional programming and domain specific modelling we have acquired funding for a project called GRACeFUL from the FETPROACT-1-2014 Global Systems Science call in Horizon 2020. The project has just started and can complement the current application, especially the case study on Domain Specific Modelling.

References

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 K. Claessen. Efficient divide-and-conquer parsing of practical context-free languages. In *Proc. of ICFP 2013*, pages 111–122, 2013.

J.-P. Bernardy and G. Moulin. A computational interpretation of parametricity. In *LICS*. IEEE, 2012.

J.-P. Bernardy and G. Moulin. Type-theory in color. In *Proc. of ICFP 2013*, pages 61–72, 2013.

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

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, 2010b. DOI:10.1017/S095679681000016X.

J.-P. Bernardy, P. Jansson, and R. Paterson. Proofs for free — parametricity for dependent types. *J. of Funct. Prog.*, 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.

N. Botta, C. Ionescu, and E. Brady. Sequential decision problems, dependently typed solutions. In *Proceedings of the Conferences on Intelligent Computer Mathematics (CICM 2013), "Programming Languages for Mechanized Mathematics Systems Workshop (PLMMS)"*, July 2013. URL ceur-ws.org/Vol-1010/paper-06.pdf.

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, 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. DOI:10.1145/1111037.1111056.

J. Duregård and P. Jansson. Embedded parser generators. In *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.

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

C. Ionescu. *Vulnerability modelling and monadic dynamical systems*. PhD thesis, Freie Universität Berlin, 2009.

C. Ionescu. Vulnerability modelling with functional programming and dependent types. Accepted for publication, to appear., 2014.

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)*, volume 8241 of *LNCS*, pages 140–156. Springer-Verlag, 2013a. DOI:10.1007/978-3-642-41582-1_9.

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. DOI:10.1145/263699.263763.

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. DOI:10.1016/S0167-6423(01)00020-X.

P. Jansson, J. Jeuring, and students of the Utrecht U. 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.

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. DOI:10.1007/978-3-642-03034-5_12.

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

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. DOI:10.1145/1411286.1411301.

N. Swamy, J. Chen, C. Fournet, P.-Y. Strub, K. Bhargavan, and J. Yang. Secure distributed programming with value-dependent types. In *Proc. of ICFP 2011*, pages 266–278, 2011.

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

L. Valiant. General context-free recognition in less than cubic time. *J. of computer and system sciences*, 10(2):308–314, 1975.

Tvärvetenskap

Min ansökan är tvärvetenskaplig

☐

Ett tvärvetenskapligt forskningsprojekt definieras i denna utlysning som att det för sitt genomförande behöver sakkunskaper, metoder, terminologi, data samt forskare från fler än ett av Vetenskapsrådets ämnesområden medicin och hälsa, naturvetenskap och teknikvetenskap, humaniora och samhällsvetenskap samt utbildningsvetenskap. Om ditt forskningsprojekt är tvärvetenskapligt enligt denna definition anger och redogör du för detta här.

Klicka här för vidare instruktioner

Vetenskaplig rapport

Vetenskaplig rapport/redogörelse för vetenskaplig verksamhet inom tidigare projekt

---

Budget och forskningsresurser

Personal i projektet

Redovisa vilken personal som ska arbeta i projektet och lönekostnaderna för dessa som du ansöker om från Vetenskapsrådet. Ange hela beloppet, ej i tusental kronor.

Medverkande forskare som accepterar inbjudan att delta i ansökan visas automatiskt under Aktivitetsgrad. Observera att det kan ta några minuter innan informationen uppdateras och det kan krävas att projektledaren stänger och sen öppnar formuläret på nytt.

Aktivitetsgrad i projektet

	Roll i projektet	Namn	Procent av heltid
1	Projektledare	Patrik Jansson	20
2	Medverkande forskare	Jean-Philippe Bernardy	20
3	Medverkande forskare	Cezar Ionescu	20
4	Medverkande forskare	Jean-Philippe Bernardy	
5	Medverkande forskare	Cezar Ionescu	

Löner inklusive sociala avgifter

	Roll i projektet	Namn	Procent av lönen	2016	2017	2018	2019	Totalt
1	Övrig ej disputerad personal	Ny doktorand	80	435 000	450 000	466 000	482 000	1 833 000
2	Projektledare	Patrik Jansson	20	194 000	201 000	208 000	215 000	818 000
	Totalt			629 000	651 000	674 000	697 000	2 651 000

Övriga kostnader

Ange övriga kostnader i projektet som du ansöker om från Vetenskapsrådet. Ange hela beloppet, ej i tusental kronor.

Lokaler

	Typ av lokal	2016	2017	2018	2019	Totalt
1	Kontor	44 000	46 000	47 000	49 000	186 000
	Totalt	44 000	46 000	47 000	49 000	186 000

Driftskostnader

	Driftskostnader	Beskrivning	2016	2017	2018	2019	Totalt
1	Direct IT costs		15 000	16 000	16 000	17 000	64 000
2	Lic & Disp.		15 000		50 000		65 000
3	Other equipment	computer	15 000			15 000	30 000
4	Travel	and conferences	40 000	40 000	40 000	40 000	160 000
	Totalt		85 000	56 000	106 000	72 000	319 000

Avskrivningar utrustning

	Avskrivning	Beskrivning	2016	2017	2018	2019

Total kostnad för projektet

Nedan summeras de kostnader som du har fört in i din budget och därmed ansöker om från Vetenskapsrådet. Indirekta kostnader för du in separat i tabellen.

Under Annan kostnad anger du eventuella ytterligare kostnader, utöver det du ansöker om från Vetenskapsrådet, som genomförandet av projektet medför. Ange hela beloppet, ej i tusental kronor.

Delsumma plus indirekta kostnader per år utgör sökt belopp från Vetenskapsrådet.

Total kostnad för projektet

Specificerade ko..	2016	2017	2018	2019	Totalt, sökt	Annan kostnad	Total kostnad
Löner inkl. sociala avgifter	629 000	651 000	674 000	697 000	2 651 000		2 651 000
Driftskostnader	85 000	56 000	106 000	72 000	319 000		319 000
Avskrivningar utrustning					0		0
Lokaler	44 000	46 000	47 000	49 000	186 000		186 000
Delsumma	758 000	753 000	827 000	818 000	3 156 000	0	3 156 000
Indirekta kostnader	239 000	247 000	256 000	265 000	1 007 000		1 007 000
Total projektkostnad	997 000	1 000 000	1 083 000	1 083 000	4 163 000	0	4 163 000

Motivering av sökt budget

Motivera kort varje sökt kostnad i budgeten.

Motivering av sökt 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 conferences, travel, computers, premises, IT-costs, licentiate and PhD defence costs.

Annan finansiering

Ange forskningsresurser för projektet under perioden i form av övrig finansiering (sökt eller redan beviljad) utöver det som söks från Vetenskapsrådet. Ange hela beloppet, ej i tusental kronor.

Annan finansiering för detta projekt

Finansiär	Sökande/projektledare	Typ av bidrag	Dnr eller motsv.	2016	2017	2018	2019

CV

B Curricula Vitæ

Curriculum Vitæ:
Patrik Jansson, 1972-03-11

1. Higher education degree:

1995: BSc+MSc degrees in Engineering Physics + 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. 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 Associate Professor:

2004: Docent degree, Chalmers, Sweden.

5. Current Employment:

2011–now: Professor, Chalmers. Research 50% (2015).

6. Previous Employment and Education:

2001–2004: Assistant Professor in Computer Science, 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 for the BSc and MSc education at the Computer Science department. On average 35% / year.

2005–2008: Vice head (of the Computer Science and Engineering department) responsible for the BSc and MSc education. On average 50% of full time / year.

2011–2013: Head of the 5-year education programme in Computer Science and Engineering (Civilingenjör Datateknik, Chalmers). On average 42% / year.

2013–: Head of the Software Technology division, Chalmers and GU. Around 45% / year.

8. Supervision experience:

I was PhD advisor of Ulf Norell (PhD 2007), Nils Anders Danielsson (PhD 2007) and Jean-Philippe Bernardy (PhD 2011). All three are still in academia.

I have also supervised four PostDocs: G. Hamon and A. Abel (2003–2005), JP. Bernardy (2011–2012) and C. Ionescu (2013–).

I currently supervise the PhD student Jonas Duregård (Lic. 2012). I am also examiner (but not supervisor) of three other PhD students: D. Rosén, A. Ekblad and M. Aronsson.

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. Grants:

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 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.

2011–2014: Main applicant on the project *Strongly Typed Libraries for Programs and Proofs* funded with **2.4M SEK** by VR.

2015–2018: Co-applicant and work package leader on the EU project “*GRACeFUL: Global systems Rapid Assessment tools through Constraint FUnctional Languages*” (FETPROACT-1-2014, **2.4M EUR**).

10. Awards, 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.

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

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

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

2011–2012: WGP Steering Committee Chair.

2012: Workshops chair of the International Conference on Functional Programming

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 M Rasetti, M Resch and R Dum.

2013: Workshops chair of 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). The call **FETPROACT-1-2014** Global Systems Science in Horizon 2020 is concrete evidence on the success of this line of work.

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.

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 have obtained my Docent degree in May 2014

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 90 work days of parental leave during my employment as Assistant Professor.

8. Supervised PhD and PostDoc:

I am currently co-supervising Guilhem Moulin (main supervisor Peter Dybjer) and Dan Rosén (main supervisor Koen Claessen).

9. Awards, grants, etc.:

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

10. Other information:

Research Statement: My interests lie in fields that contribute to bridge the gap between abstraction and efficiency, including:

- constructive type theory
- functional programming
- generic programming
- software engineering

I have more specifically pursued two independent line of research:

- The first topic borders functional programming and language technology. At the start of my PhD studies, I investigated syntax-driven feedback for interactive programming environments. This led to research on incremental, parallel and robust parsing, yielding three Master’s theses and three publications.
- The second topic borders functional programming and type-theory, more precisely bringing in more and more type-theory into the realm of functional programming. In particular, the main topic of my PhD is parametricity: a technique which enables functional programmers to leverage polymorphic types to ensure correctness of programs. Besides my PhD thesis, this line of research has led to five publications.

  Recently, as part of the SSF-Funded Resource-Aware Functional Programming project, I am working on applying the methodology of type-theory to the

domain of low-level functional programming. Indeed, functional programming languages with linear types have long held the promise to be both low-level and allow higher-order abstractions, but are not yet very well known in my research community. I have contributed to spreading the ideas by leading a course in the computer science graduate school.

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 (2012–2014).
- 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 (Proof assistant and dependently-typed language)

Community roles:

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

Curriculum Vitæ:
Cezar Ionescu, 19681221-1479

1. Higher education degree:

1988–1993: Politehnica University of Bucharest, Faculty of Control Engineering and Computer Science, specialization Bioinformatics. Thesis on *Hardware Implementation of Neural Networks*, awarded the highest grade, 10.

2. Doctoral degree:

2009: PhD from the Fachbereich Informatik and Mathematik of the Freie Universität Berlin with the thesis *Vulnerability Modeling and Monadic Dynamical Systems* (summa cum laude). Advisor: Prof. Rupert Klein.

3. PostDoc:

2009–2013: Postdoc at the Potsdam Institute for Climate Impact Research within the Research Domain *Transdisciplinary Concepts and Methods*. Special focus on using type theory to specify economic models.

4. Docent degree:

None.

5. Current Employment:

since August 2013: Postdoc at Chalmers. Work on *increasingly correct scientific programming*, using type theory and validated numerical methods. Research 80%.

6. Prev. Employment:

- 2009–2013: Postdoc position at PIK, within the project *Model Specification and Program Development*. Focus on using dependently-typed programming for specifiying, developing, testing, extending and re-factoring implementations of economic and multi-agent models used at PIK.
- 2006–2009: Scientific position at PIK. Worked within the project FAVAIA (*Formal Approaches to Vulnerability, Adaptation and Integrated Assessment*), part of the EU project *Adaptation and Mitigation (ADAM)*. Formalized the concept of "vulnerability" as used in the global change community and completed PhD thesis on vulnerability. Continued work on IT-related projects, among them the PIK project S (*Software components for distributed adaptive finite volume methods*), where I formulated a mathematical model for a class of parallel programs and assisted in the specification of relation-based algorithms.
- 1999–2006: IT position at the Potsdam Institute for Climate Impact Research (PIK), project Modenv (*Modeling Environment*). Implemented a Fortran code analysis tool for coupling legacy models with Corba, a simulation builder application for remote distributed components (*Graphical Simulation Builder*), participated in the design and implementation of the *Typed Data Transfer* library and was responsible for the Python version.
- 1993–1999: Systems analyst at the Informatics Research Institute in Bucharest. Worked within the Artificial Intelligence laboratory on projects related to optimization and adaptive control using neural networks, fuzzy logic and genetic algorithms. In 1998 worked on a Y2K project for Cap Gemini Nederland where I implemented several tools for analyzing COBOL code and assisting programmers in the removal of Y2K-related bugs.

7. Interruptions in research:

I have taken six months of parental leave during 2007–2008.

8. Supervised PhD and PostDoc:

I have co-supervised the following PhD students:

- Sarah Wolf (main supervisor Rupert Klein). PhD in mathematics, Freie Universität Berlin, 2010. Thesis: *From Vulnerability Formalization to Finitely Additive Probability Monads*.
- Daniel Lincke (main supervisor Sibylle Schupp). PhD in computer science, Technische Universität Hamburg-Harburg, 2012. Thesis: *A transformational approach to generic software development based on higher-order, typed functional signatures.*

9. Other information:

Project proposals:

- Co-writer of the proposal *GRACeFUL - Global systems Rapid Assessment tools through Constraint FUnctional Languages*, accepted within the FETPROACT1-2014 call of the EU Horizon 2020 program. The project has started on 2014-02-01 and will last for three years.
- Co-writer of the proposal for the Coordinated Action *Global Systems Dynamics and Policy* (2010–2012).
- Co-writer of the proposal for the *Adaptation and Mitigation* EU Project (2006–2009), specifically the proposal to formalize the cluster of notions related to "vulnerability to climate change".

Teaching:

- Currently developing, together with Patrik Jansson, a BSc level course on *Domain-Specific Languages of Mathematics*. The description of the course can be found at

  `http://tinyurl.com/q7ns93m`
- *Category Theory for Functional Programming*, winter semester 2014-2015, Chalmers. Graduate course. Course notes will be integrated in the volume *Mathematics for Scientific Computing*, to be published in the Oberwolfach Seminar Series by Birkhäuser. Course description can be found at

  `http://tinyurl.com/ozh8pce`
- *Introduction to Programming using Java*, winter semester 2014-2015, University of Göteborg. First-year course. Course materials can be found at

  `http://tinyurl.com/q35g5cj`
- *Introduction to Programming using Java*, winter semester 2013-2014, University of Göteborg. First-year course. Course materials can be found at

  `http://tinyurl.com/q4xb93y`
- *Advanced Functional Programming*, summer semester 2012-2013, Freie Universität Berlin. Masters level course. Course materials can be found at

  `http://tinyurl.com/pxc5uxk`

Other:

- Co-organizer of the Oberwolfach Seminar *Mathematics for Scientific Programming*, November 2013.
- Invited observer to IFIP Working Group 2.1 on *Algorithmic Languages and Calculi*, attended meetings 68 (February 2012, Rome, Italy) and 71 (March 2014, Zeegse, the Netherlands).
- Co-founder of the *Cartesian Seminar*, a weekly scientific seminar held at PIK (2000–2012), and since 2012 at the Potsdam University, with the aim of creating a common understanding of scientific issues in an interdisciplinary (but mathematically inclined) audience, based on close reading of essential texts and congenial dialog.
- Chairman at the Dahlem conference *New Approaches in Economics after the Financial Crisis* (August 28 to 31, 2010).
- Invited presentation at the Dahlem conference *Is There a Mathematics of Social Entities?* (December 14 to 19, 2008) on *Modeling versus Formalization*.
- Rapporteur for Group 4 *Models, Metaphors, and Visualisation* at the Dahlem conference *Is There a Mathematics of Social Entities?* (December 14 to 19, 2008).

Publikationer

C Publication lists

⇒ An arrow on the left marks the publications most relevant for this project.

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.

0. Most cited publications (Google Scholar, 2015-03-28)

Jansson's Hirsch-index is **18**, his total citation count is over **1400** and the following papers are the five most cited.

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: **331**.

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. URL http://www.cse.chalmers.se/~patrikj/poly/afp98/.
   Number of citations: **197**.

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. URL http://www.cse.chalmers.se/~patrikj/poly/AFP96.pdf.
   Number of citations: **162**.

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: **67**.

5. 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. DOI:10.1145/1111037.1111056.
   Number of citations: **60**.

1. Journal articles (last 8 years)

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: **21**.

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, 2010c. DOI:10.1017/S095679681000016X.
   Number of citations: **15**.

⇒ 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: **13**.

2. Refereed conference articles (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)*, volume 8241 of *LNCS*, pages 140–156. Springer-Verlag, 2013a. DOI:10.1007/978-3-642-41582-1_9.
Number of citations: **2**.

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, 2013b. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. DOI:10.4230/LIPIcs.TYPES.2011.41.
Number of citations: **3**.

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: **17**.

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: **4**.

13. J. Duregård and P. Jansson. Embedded parser generators. In *Haskell '11*, pages 107–117, New York, NY, USA, 2011. ACM. DOI:10.1145/2034675.2034689.
Number of citations: **8**.

⇒ 14. J.-P. Bernardy, P. Jansson, and R. Paterson. Parametricity and dependent types. In *Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*, pages 345–356, Baltimore, Maryland, 2010b. ACM. DOI:10.1145/1863543.1863592.
Number of citations: **37**.

⇒ 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, 2010a. DOI:10.1007/978-3-642-11957-6_8.
Number of citations: **23**.

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: **66**.

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: **7**.

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, 2008a. 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: **13**.

20. P. Jansson, J. Jeuring, and students of the Utrecht U. 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: **6**.

3+4. Review articles, book chapters, books

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

22. 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 2013b.

23. 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.

24. 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. URL http://www.cse.chalmers.se/~patrikj/papers/IMPACT_SEFI_2009_final.pdf.

25. 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.

26. 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.

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

5. Patents

None.

6. 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),

28. U. Norell et al. Agda — a dependently typed programming language. Implementation available from `http://wiki.portal.chalmers.se/agda/`, 2008.
    Number of citations: **428**.

    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.

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

29. P. Jansson and J. Jeuring. The haskell package *ClassLaws*. `http://hackage.haskell.org/package/ClassLaws`, 2012. (690 downloads)

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`. (3342 downloads)
- 2006: The BNF Converter — a tool for generating a compiler skeleton from a labelled BNF grammar: `http://bnfc.digitalgrammars.com/`
- 2010: BNFC-meta: a library for embedded parser generators: `https://hackage.haskell.org/package/BNFC-meta` (3193 downloads)
- 2012: testing-feat: a library for efficiently enumerating large abstract syntax tree types: `http://hackage.haskell.org/package/testing-feat`. (7531 downloads)
- My github profile (`https://github.com/patrikja`) collects most of my Open Source contributions.

7. Popular science articles/presentations

30. 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. `http://wiki.portal.chalmers.se/cse/pmwiki.php/FP/SustainableEnergyFuture`, 2013.

Invited presentations (2007–2014)

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.

2014-03-09: “An algebra for parallel parsing” at the IFIP Working Group 2.1 on Algorithmic Languages and Calculi, meeting #71 in Zeegse, the Netherlands.

2014-10-10: “Domain-Specific Languages for Global Systems Science” at the Third Open Global Systems Science Conference in Brussels, Belgium.

Selected Publications: Jean-Philippe Bernardy

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 *LICS* and *ICFP* below) typically have the highest impact factor within that field, higher even than any journal.

C.1 Peer-reviewed publications in journal

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: **14**.

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, 2010c. DOI:10.1017/S095679681000016X.
Number of citations: **13**.

C.2 Peer-reviewed publications in conferences and workshops

J.-P. Bernardy and K. Claessen. Efficient divide-and-conquer parsing of practical context-free languages. In *Proceedings of the 18th ACM SIGPLAN international conference on Functional Programming*, pages 111–122, 2013.
Number of citations: **0**.

J.-P. Bernardy and N. Pouillard. Names for free — polymorphic views of names and binders. In *Proceedings of the 6th ACM SIGPLAN symposium on Haskell*, pages 13–24. ACM, 2013.
Number of citations: **1**.

J.-P. Bernardy and G. Moulin. Type-theory in color. In *Proceedings of the 18th ACM SIGPLAN international conference on Functional Programming*, pages 61–72, 2013.
Number of citations: **4**.

J.-P. Bernardy and G. Moulin. A computational interpretation of parametricity. In *LICS*. IEEE Computer Society, 2012.
Number of citations: **7**.

J.-P. Bernardy and M. Lasson. Realizability and parametricity in pure type systems. In M. Hofmann, editor, *Foundations Of Software Science And Computational Structures*, volume 6604 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2011.
Number of citations: **16**.

J.-P. Bernardy, P. Jansson, and R. Paterson. Parametricity and dependent types. In *Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*, pages 345–356, Baltimore, Maryland, 2010b. ACM. DOI:10.1145/1863543.1863592.
Number of citations: **33**.

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, 2010a. DOI:10.1007/978-3-642-11957-6_8.
Number of citations: **17**.

J.-P. Bernardy. Lazy functional incremental parsing. In *Proceedings of the 2nd ACM SIGPLAN symposium on Haskell*, pages 49–60, Edinburgh, Scotland, 2009. ACM. ISBN 978-1-60558-508-6. DOI:10.1145/1596638.1596645. URL http://portal.acm.org/citation.cfm?

id=1596638.1596645.
Number of citations: **10**.

J.-P. Bernardy, P. Jansson, M. Zalewski, S. Schupp, and A. Priesnitz. A comparison of C++ concepts and Haskell type classes. In *WGP '08: Proceedings of the ACM SIGPLAN workshop on Generic programming*, pages 37–48, Victoria, BC, Canada, 2008b. ACM. ISBN 978-1-60558-060-9. URL http://dx.doi.org/10.1145/1411318.1411324.
Number of citations: **19**.

C.3 Non peer-reviewed publications

J.-P. Bernardy. Yi: an editor in Haskell for Haskell. In *Proceedings of the first ACM SIGPLAN symposium on Haskell*, pages 61–62, Victoria, BC, Canada, 2008. ACM. ISBN 978-1-60558-064-7. DOI:10.1145/1411286.1411294. URL http://portal.acm.org/citation.cfm?id=1411286.1411294.
Number of citations: **7**.

Publicly available implementations

I am the main contributor to the Yi project
J.-P. Bernardy. Yi: an editor in Haskell for Haskell. In *Proceedings of the first ACM SIGPLAN symposium on Haskell*, pages 61–62, Victoria, BC, Canada, 2008. ACM. ISBN 978-1-60558-064-7. DOI:10.1145/1411286.1411294. URL http://portal.acm.org/citation.cfm?id=1411286.1411294.

I have contributed the Agda proof assistant. 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.

Selected Publications: Cezar Ionescu

0. Most cited publications (Google Scholar, 2015-03-30)

⇒ C. Ionescu, R. J. T. Klein, J. Hinkel, K. S. Kavi Kumar, and R. Klein. Towards a formal framework of vulnerability to climate change. *Environmental Modelling and Assessment*, 14(1):1–16, 2009.
Number of citations: **172**.

T. Downing, J. Aerts, J. Soussan, O. Barthelemy, S. Bharwani, C. Ionescu, J. Hinkel, R. Klein, L. Mata, N. Martin, et al. Integrating social vulnerability into water management. Technical report, NeWater Working Paper, 2005.
Number of citations: **63**.

⇒ C. Ionescu. *Vulnerability modelling and monadic dynamical systems*. PhD thesis, Freie Universität Berlin, 2009.
Number of citations: **20**.

⇒ J. Hinkel, D. Lincke, A. T. Vafeidis, M. Perrette, R. J. Nicholls, R. S. Tol, B. Marzeion, X. Fettweis, C. Ionescu, and A. Levermann. Coastal flood damage and adaptation costs under 21st century sea-level rise. *Proceedings of the National Academy of Sciences*, 111(9):3292–3297, 2014.
Number of citations: **17**.

⇒ S. Wolf, J. Hinkel, M. Hallier, A. Bisaro, D. Lincke, C. Ionescu, and R. J. Klein. Clarifying vulnerability definitions and assessments using formalisation. *International Journal of Climate Change Strategies and Management*, 5(1):54–70, 2013.
Number of citations: **15**.

1. Peer-reviewed publications in journal

⇒ C. Ionescu. Vulnerability modelling with functional programming and dependent types. *Mathematical Structures in Computer Science*, FirstView:1–15, 12 2014. URL `http://journals.cambridge.org/article_S0960129514000139`.
Number of citations: **0**.

⇒ J. Hinkel, D. Lincke, A. T. Vafeidis, M. Perrette, R. J. Nicholls, R. S. Tol, B. Marzeion, X. Fettweis, C. Ionescu, and A. Levermann. Coastal flood damage and adaptation costs under 21st century sea-level rise. *Proceedings of the National Academy of Sciences*, 111(9):3292–3297, 2014.
Number of citations: **17**.

⇒ D. Lincke, S. Schupp, and C. Ionescu. Functional prototypes for generic c++ libraries: a transformational approach based on higher-order, typed signatures. *International Journal on Software Tools for Technology Transfer*, pages 1–15, 2014.
Number of citations: **0**.

⇒ N. Botta, A. Mandel, C. Ionescu, M. Hofmann, D. Lincke, S. Schupp, and C. Jaeger. A functional framework for agent-based models of exchange. *Applied Mathematics and Computation*, 218 (8):4025 – 4040, 2011.
Number of citations: **6**.

⇒ C. Ionescu, R. J. T. Klein, J. Hinkel, K. S. Kavi Kumar, and R. Klein. Towards a formal framework of vulnerability to climate change. *Environmental Modelling and Assessment*, 14(1):1–16, 2009.
Number of citations: **172**.

N. Botta and C. Ionescu. Relation-based computations in a monadic BSP model. *Parallel Computing*, 33(12):795 – 821, 2007.
Number of citations: **12**.

2. Peer-reviewed publications in conferences and workshops

⇒ N. Botta, A. Mandel, M. Hofmann, S. Schupp, and C. Ionescu. Mathematical specification of an agend-based model of exchange. In *Enabling Domain Experts to use Formalised Reasoning*, page NA, 2013b.
Number of citations: **2**.

⇒ 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: **3**.

⇒ 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)*, volume 8241 of *LNCS*, pages 140–156. Springer-Verlag, 2013a. DOI:10.1007/978-3-642-41582-1_9.
Number of citations: **2**.

⇒ N. Botta, C. Ionescu, and E. Brady. Sequential decision problems, dependently typed solutions. In *Proceedings of the Conferences on Intelligent Computer Mathematics (CICM 2013), "Programming Languages for Mechanized Mathematics Systems Workshop (PLMMS)"*, July 2013a. URL ceur-ws.org/Vol-1010/paper-06.pdf.
Number of citations: **1**.

3. Review articles

⇒ C. C. Jaeger, L. Paroussos, D. Mangalagiu, R. Kupers, A. Mandel, J. D. Tàbara, N. Botta, S. Fürst, E. Henning, C. Ionescu, et al. A new growth path for europe. *Generating Prosperity and Jobs in the Low-Carbon Economy. Synthesis Report PIK, University of Oxford, ICCS, Université Paris*, 1, 2011.
Number of citations: **15**.

4. Books

⇒ C. Ionescu. *Vulnerability modelling and monadic dynamical systems*. PhD thesis, Freie Universität Berlin, 2009.
Number of citations: **20**.

5–6. Patents, open-access computer programs

None.

7. Popular science articles/presentations

S. Wolf, J. Hinkel, M. Hallier, A. Bisaro, D. Lincke, C. Ionescu, and R. J. Klein. Clarifying vulnerability definitions and assessments using formalisation. *International Journal of Climate Change Strategies and Management*, 5(1):54–70, 2013.
Number of citations: **15**.

CV

Namn: Patrik Jansson	**Dr-examen:** 2000-06-09
Födelsedatum: 19720311	**Akademisk titel:** Annan
Kön: Man	**Arbetsgivare:** Chalmers tekniska högskola

Forskarutbildning

Avhandlingens titel (sv)
Funktionell polytypisk programmering

Avhandlingens titel (en)
Functional Polytypic Programming

Organisation	Enhet	Handledare
Chalmers tekniska högskola, Sverige Sverige - Universitet och högskolor	Inst för Data- och informationsteknik	Johan Jeuring

Disputationsämne	ISSN/ISBN-nummer	Doktorsexamensdatum
10201. Datavetenskap (datalogi)	91-7197-895-X	2000-06-09

CV

Namn: Jean-Philippe Bernardy	**Dr-examen:** 2011-06-07
Födelsedatum: 19781215	**Akademisk titel:** Docent
Kön: Man	**Arbetsgivare:** Ingen nuvarande arbetsgivare

Forskarutbildning

Avhandlingens titel (sv)

Avhandlingens titel (en)

Organisation	Enhet	Handledare
Chalmers tekniska högskola, Sverige Sverige - Universitet och högskolor	3725 - Software Technology	

Disputationsämne	ISSN/ISBN-nummer	Doktorsexamensdatum
10201. Datavetenskap (datalogi)		2011-06-07

CV

Namn: Cezar Ionescu	**Dr-examen:** 2009-02-09
Födelsedatum: 19681221	**Akademisk titel:** Doktor
Kön: Man	**Arbetsgivare:** Chalmers tekniska högskola

Forskarutbildning

Avhandlingens titel (sv)

Avhandlingens titel (en)

Vulnerability Modeling and Monadic Dynamical Systems

Organisation	Enhet	Handledare
Freie Universität Berlin, Tyskland Ej Sverige - Universitet och högskolor	Fachbereich Mathematik und Informatik	Rupert Klein

Disputationsämne	ISSN/ISBN-nummer	Doktorsexamensdatum
10103. Algebra och logik		2009-02-09

Publikationer

Namn: Patrik Jansson	**Dr-examen:** 2000-06-09
Födelsedatum: 19720311	**Akademisk titel:** Annan
Kön: Man	**Arbetsgivare:** Chalmers tekniska högskola

Jansson, Patrik har inte lagt till några publikationer till ansökan.

Publikationer

Namn: Jean-Philippe Bernardy	**Dr-examen:** 2011-06-07
Födelsedatum: 19781215	**Akademisk titel:** Docent
Kön: Man	**Arbetsgivare:** Ingen nuvarande arbetsgivare

Bernardy, Jean-Philippe har inte lagt till några publikationer till ansökan.

Publikationer

Namn: Cezar Ionescu	**Dr-examen:** 2009-02-09
Födelsedatum: 19681221	**Akademisk titel:** Doktor
Kön: Man	**Arbetsgivare:** Chalmers tekniska högskola

Ionescu, Cezar har inte lagt till några publikationer till ansökan.

Registrera

Villkor

Ansökan ska förutom av den sökande även signeras av behörig företrädare för medelsförvaltaren. Företrädaren är vanligtvis prefekten vid den institution där forskningen ska bedrivas, men ska i vissa fall utgöras av exempelvis rektor. Detta framgår i sådana fall av den aktuella utlysningstexten för bidraget.

Signering av *den sökande* innebär en bekräftelse av att:

- uppgifterna i ansökan är korrekta och följer Vetenskapsrådets instruktioner
- bisysslor och kommersiella bindningar har redovisats för medelsförvaltaren och att det där inte framkommit något som strider mot god forskningssed
- nödvändiga tillstånd och godkännanden finns vid projektstart, exempelvis avseende etikprövning.

Signering av *medelsförvaltaren* innebär en bekräftelse av att:

- den beskrivna forskningen, anställningen och utrustningen kan beredas plats inom institutionen under den tid och i den omfattning som anges i ansökan
- institutionen godkänner kostnadsberäkningen i ansökan
- projektet bedrivs i enlighet med svensk lagstiftning.

Ovanstående punkter ska ha diskuterats mellan parterna innan företrädaren för medelsförvaltaren godkänner och signerar ansökan.

Projektskisser ska ej signeras av medelsförvaltaren. Medelsförvaltaren ska endast signera den fullständiga ansökan om skissen går vidare till steg två.

Ansökningar där en organisation är sökande signeras automatiskt vid registrering av ansökan.