VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kansliets noteringar Kod	Dnr
2013-40060-105052-38	2013-4879

2013
Project Grant Junior Researchers

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
698	1108	1222	1254	1399

APPLICANT

Name(Last name, First name)
Danielsson, Nils Anders

Date of birth
791006-6013

Gender
Male

Email address
nad@cse.gu.se

Academic title
PhD (engi)

Position
Assistant professor

Phone
031-7721680

Doctoral degree awarded (yyyy-mm-dd)
2007-12-05

WORKING ADDRESS

University/corresponding, Department, Section/Unit, Address, etc.
Göteborgs universitet
Institutionen för data- och informationsteknik
Datavetenskap

41296 Göteborg, Sweden

ADMINISTRATING ORGANISATION

Administrating Organisation
Göteborgs universitet

DESCRIPTIVE DATA

Project title, Swedish (max 200 char)
Praktisk nästlad induktion och koinduktion

Project title, English (max 200 char)
Practical nested induction and coinduction

Abstract (max 1500 char)
In computer science there is often a need to model phenomena that are partly infinite and partly finite. Examples include liveness properties, equivalence of concurrent processes (weak bisimilarity), and subtyping for recursive types. A natural way to model such phenomena is to use so-called coinduction for the infinite parts, and induction for the finite ones.

Infinite phenomena, and especially phenomena that are partly infinite and partly finite, can have surprising properties. To illustrate this point one can note that Robin Milner, a Turing Award winner, has published an incorrect result about weak bisimilarity. To avoid mistakes we can work in a formal setting, letting tools check our proofs for us. If we use total, dependently typed programming languages, then we can write specifications, proofs and programs in an integrated setting, and some of these languages have features that allow us to use complicated (nested) combinations of induction and coinduction. However, currently these features can be quite cumbersome to use.

This project has three main parts:

* Find a nice design that allows us to combine induction and coinduction in a convenient, practically useful way.

* Incorporate this design into the (fairly popular) dependently typed programming language Agda.

* Make use of the design to explore new applications of nested induction and coinduction.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2013-40060-105052-38

Name of Applicant
Danielsson, Nils Anders

Date of birth
791006-6013

Abstract language
English

Keywords
induction, coinduction, dependent types

Research areas
*Nat-Tek generellt

Review panel
NT-2, NT-1

Classification codes (SCB) in order of priority
10201

Aspects

Application is also submitted to

similar to:

identical to:

ANIMAL STUDIES

Animal studies
No animal experiments

OTHER CO-WORKER

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)

Name(Last name, First name)
,

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

Date of birth

Gender

Academic title

Doctoral degree awarded (yyyy-mm-dd)

Kod
2013-40060-105052-38

Name of Applicant
Danielsson, Nils Anders

Date of birth
791006-6013

ENCLOSED APPENDICES

A, B, C, N, S

APPLIED FUNDING: THIS APPLICATION

Funding period (planned start and end date)
2014-01-01 -- 2018-12-31

Staff/ salaries (kSEK)

Main applicant	% of full time in the project	2014	2015	2016	2017	2018
Nils Anders Danielsson	40		430	443	456	469
Other staff						
PhD student	80	577	581	638	698	745
Total, salaries (kSEK):		577	1011	1081	1154	1214

Other projectrelated costs (kSek)	2014	2015	2016	2017	2018
Computer equipment and literature	26	1	20	1	21
Travel and conferences	95	96	108	99	144
Publication costs			13		20
Total, other costs (kSEK):	121	97	141	100	185

Total amount for which applied (kSEK)

2014	2015	2016	2017	2018
698	1108	1222	1254	1399

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
2012-5294	3000	3000	

Project title	Applicant
Types for proofs and programs	Thierry Coquand (Funding 2015-2016: 3000 kSEK/year.)

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

Funding source	Total	Proj.period	Applied 2014
ERC	16000	2010-2015	

Project title	Applicant
Formalization of Constructive Mathematics	Thierry Coquand

POPULAR SCIENCE DESCRIPTION

Popularscience heading and description (max 4500 char)
Ändliga och oändliga fenomen: teorier, verktyg och tillämpningar

Det långsiktiga målet med min forskning är att göra det enklare och trevligare att skriva program som inte innehåller fel (d v s inte kraschar, inte ger fel resultat, o s v). Program och matematiska bevis är ganska lika, så målet gäller även matematiska bevis.

Ibland när man skriver ett program (eller modellerar ett system, eller bevisar ett teorem) vill man hantera begrepp eller egenskaper som i någon mån är "oändliga". Ett exempel kan vara ett program som hanterar antisladdsystemet i en bil: så länge ingen stänger av bilen ska antisladdsystemet vara aktivt. Man kan tänkas vilja bevisa egenskaper som att "det /alltid/ kommer att vara fallet att, om bilen är igång och sladdar, så kommer sladdrörelsen att kompenseras".

Det här projektet fokuserar på en bra, formell hantering av fenomen som har både ändliga och oändliga drag. Ett exempel från min egen forskning: Jag har utvecklat en formalism för att uttrycka en mycket stor klass av formella språk. Den här formalismen beskriver språk som ett slags matematiska "träd" som kan vara oändligt höga. För att se till att man alltid kan tolka meningar uttryckta i de här språken (upp till eventuell tvetydighet) får dock träden inte förgrena sig alltför snabbt: ändlig förgrening går att hantera, men vissa former av oändlig förgrening är förbjudna.

Jag har använt ett programmeringsspråk/bevissystem vid namn Agda för att definiera den här klassen av språk. Agda har ganska bra stöd för modellering av fenomen som är både oändliga och ändliga på samma gång. Man kan till exempel ge en definition av en så kallad datatyp som bara tillåter korrekt förgrenade träd. Om programmeraren sedan försöker definiera ett träd som inte uppfyller kraven så klagar Agda. Det går också att använda Agda för att bevisa matematiska teorem, t ex att det alltid går att tolka meningar uttryckta i ett av "trädspråken".

Agda är dock inte perfekt. Det finns flera problem som gör det onödigt krångligt att arbeta med oändliga fenomen. Planen är att ta fram en design för ett litet programmeringsspråk som undviker några av problemen, använda den här designen för att förbättra Agda, och dessutom använda Agda för att utforska fler fenomen med både ändliga och oändliga drag.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
þÿ2013-40060-10 0 2-

Name of applicant
Danielsson, Nils Anders

Date of birth
791006-6013

Title of research programme
Practical nested induction and coinduction

Appendix A

Research programme

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

Appendix A: Research programme

Nils Anders Danielsson

1 Purpose and aims

My overall goal is to make it easier and more pleasant to write elegant and bug-free programs and proofs. I pursue this goal through my work on dependently typed functional programming languages.

In this project I focus on complicated, partly finite and partly infinite phenomena. In certain programming languages these phenomena can be represented using "nested induction and coinduction". The project has two main goals:

- To make it easier to use nested induction and coinduction in a formal setting—in particular, in dependently typed programming languages. Here *use* refers to writing specifications, programs and proofs.
- To develop and demonstrate new applications of nested induction and coinduction.

2 Survey of the field

In computer science there is often a need to model phenomena that are partly infinite and partly finite. As a concrete example, take liveness properties, which have the form "it will always be the case that eventually something happens"; here "always" refers to something infinite, and "eventually" to something finite. Properties of this kind can be specified formally through the use of induction (finite) and coinduction (infinite).

The interaction between finite and infinite phenomena can be subtle. For instance, Robin Milner (a Turing Award winner) has published an incorrect result about weak bisimilarity for concurrent processes, as explained by Sangiorgi and Milner (1992)—and weak bisimilarity, a form of equivalence, is a prime example of a concept that has flavours of both the finite and the infinite. Nakata and Uustalu (2010) show that weak bisimilarity[1] can be expressed using *nested* induction and coinduction, a particularly complicated form of definition.

We can avoid mistakes like Milner's by working in a formal setting: when formalising proofs we can use tools that check that our arguments are correct. One class of such tools consists of the so-called *total, dependently typed functional programming languages*: these languages allow us to write specifications, programs and proofs, all in a unified setting. As an example of what can be done using such languages we have the CompCert project's formally verified compiler for a large subset of C (Leroy 2009). Some of these languages also support nested induction and coinduction.

Using such a language we can make direct use of nested induction and coinduction in programs and proofs. For instance, consider a program that reads packets from one network stream

[1] For resumptions.

and transmits processed packets to another network stream. Let us assume that the input stream is unbounded, with a finite delay between any two packets, and let us also *require* that there is a finite delay between any two packets written to the output stream. This liveness property can be enforced in a natural way by representing stream processors using a data type involving nested induction and coinduction (Hancock et al. 2009). In the language Agda (Norell 2007; Agda Team 2013) we can define a type of stream processors, *SP*, as follows (Danielsson and Altenkirch 2010):

$$\begin{aligned}&\textbf{data}\ \mathit{SP}\ :\ \mathit{Set}\ \textbf{where}\\ &\quad \mathsf{write}\ :\ \mathit{Packet} \to \infty\,\mathit{SP} \to \mathit{SP}\\ &\quad \mathsf{read}\ \ :\ (\mathit{Packet} \to \mathit{SP}) \to \mathit{SP}\end{aligned}$$

The command write p sp writes the packet p and continues as the stream processor sp, and the command read f reads a packet p and continues as the stream processor $f\ p$; f is a function that, given a packet, returns a stream processor. The use of ∞ implies that stream processors can be infinitely large—this is necessary if we want to handle unbounded streams. As a very simple example of a stream processor we have *copy*, which copies its input to its output:

$$\mathit{copy}\ =\ \mathsf{read}\ (\lambda\, p \to \mathsf{write}\, p\ (\textbf{delay}\ \mathit{copy}))$$

This processor reads one packet p, writes the packet, and starts over. Note that *copy* is a recursively defined infinite value; the **delay** construct is used to ensure that the processor is only unfolded on demand (lazily). Agda checks that infinite values are *productive*: in the case of *SP* it means that the next constructor, write or read, is always computed in finite time. This is not enough to guarantee finite delay between packets in the output stream, because we could read input packets forever and never write anything. However, the fact that ∞ is used only in the definition of write and not in the definition of read implies that one cannot have an infinite sequence of read commands without any intervening writes—this is also checked by the Agda implementation—and hence we get the required liveness guarantee (given the assumption that the Agda system is bug-free). The use of ∞ only for one constructor in the definition of *SP* is an example of nested induction and coinduction.

As mentioned above several dependently typed languages have some support for nested induction and coinduction. The most mature systems may be Coq (Coq Development Team 2012) and Agda, but both come with some form of drawback:

- Coq's support for coinduction is arguably broken, because the type system does not have the property of subject reduction, or preservation of types (Giménez 1996). Coq supports nested induction and coinduction via an encoding (Nakata and Uustalu 2010).

- Agda directly supports the nesting of induction inside coinduction, as in the example above, but not the other way around (Altenkirch and Danielsson 2010); Setzer (2010) suggests a partial solution.

Both systems employ a syntactic restriction, guarded corecursion (Coquand 1994), to ensure that infinite values can be computed productively. This restriction breaks modularity and can be rather awkward in practice (Danielsson 2010b). The experimental prototype MiniAgda (Abel 2010, 2012) supports nested induction and coinduction and uses more flexible sized types (Hughes et al. 1996; Barthe et al. 2004; Abel 2009; Abel and Pientka 2013) instead of guarded corecursion, but needs more work to be convenient to use in practice. All these languages have the problem that the natural equality for coinductive structures—bisimilarity—does not in general allow us to substitute equals for equals; McBride (2009) proposes a solution. Another solution is provided by a definitional package for Isabelle/HOL (Traytel et al. 2012); however, HOL is not a dependently typed language, and the Isabelle code generator only guarantees partial correctness, not total correctness.

To summarise, I know of no design for a dependently typed programming language that satisfies all of the following criteria:

1. Direct support for nested induction and coinduction.
2. A modular mechanism for ensuring that programs are productive.
3. Subject reduction.
4. Bisimilarity as the substitutive equality for coinductive types: “substitution of bisimilars for bisimilars”.

Despite the limitations of current systems people have used them for formalisations and programs involving nested induction and coinduction:

- Giménez (1996) uses nested induction and coinduction in Coq to represent a process calculus, and specifies, implements and verifies a simulator for the processes.
- Danielsson and Altenkirch (2010) use nested induction and coinduction in Agda to give a new definition of subtyping for recursive types, prove that it is equivalent to other definitions, and implement a formally correct decision procedure for subtyping. Komendantsky (2012) defines subtyping for recursive types in a very similar way in Coq.
- Nakata and Uustalu (2010) use nested induction and coinduction in Coq to define weak bisimilarity for resumptions as well as several semantics for a While language. Danielsson (2012b) uses nested induction and coinduction in Agda to define several notions of weak bisimilarity, and uses these to state and formally prove type soundness and compiler correctness results. Im et al. (2013) use nested induction and coinduction in Coq to define type equivalence and type contractiveness, both used in a type system with recursive, parametrised, and abstract types, and prove type soundness.
- Danielsson (2010a) defines an embedded grammar language (“parser combinators”) in Agda. Using a careful combination of dependent types and nested induction and coinduction he ensures that language membership is always decidable for these grammars, which can be infinitely large and can represent any language that can be decided using Agda. Danielsson and Norell (2011) use a variant of this grammar language to define grammars for mixfix operators.
- Jeffrey and Rathke (2011) define a library for streaming I/O in Agda using nested induction and coinduction and identify a class of programs that run in constant space. They also state that a mechanised correctness proof caught a bug leading to “subtle buffering errors”, and claim that this bug would be hard to catch using unit tests.
- Nakata et al. (2011) discuss properties of infinite streams and trees, including some properties that are defined using nested induction and coinduction in Coq.

Nested induction and coinduction has also been used without (documented) support of a tool like Agda or Coq (Park 1980; Raffalli 1994; Brandt and Henglein 1998; Levy 2006; Bradfield and Stirling 2007; Abel 2009; Hancock et al. 2009; Berger 2011; Bezem et al. 2012; Uustalu 2013); I believe that more work would be carried out formally using such tools if they were less cumbersome to use.

3 Project description

My plan is that the project will be carried out by me (using 40% of my time) and a PhD student (80%, including PhD courses and similar activities). The project consists of the following tasks:

1. *Design a small dependently typed language with good support for nested induction and coinduction.*

   The plan is that this task will be performed by me, with help from the PhD student, mainly during the first years of the project.

   The design should satisfy the criteria listed in Section 2:

   - *Direct support for nested induction and coinduction* and *a modular mechanism for ensuring that programs are productive.*

     As mentioned above MiniAgda (Abel 2010, 2012) supports nested induction and coinduction. In previous work I have identified a number of programs that it is hard to define in a modular way in Agda (Danielsson 2010b, 2012b). In the finished design it should be possible to implement most of these programs in a modular way. Abel (personal communication) and I (Danielsson 2012b) have shown that this is already possible in MiniAgda, using sized types—types annotated with bounds on the sizes of values. For this reason I plan to base the design on sized types.

     With sized types as implemented in MiniAgda the programmer needs to annotate the code with extra information to please the type-checker, even in cases where this is not necessary in Agda or Coq. My goal is to find a design that avoids the need to consider sizes in certain common cases, but allows for seamless integration of size information when necessary: I want to avoid a situation where programmers are compelled to litter the code with size information, just in case it is needed further down the line.

     More work is also needed to integrate sized types with implicit arguments, which are not available in MiniAgda. Implicit arguments as implemented in Agda allow programmers to omit much of the type information. A program that does not use implicit arguments can be much larger than one which does.

   - *Subject reduction.*

     As mentioned above subject reduction fails to hold in Coq; an unreleased version of Agda had the same problem. McBride (2009) discusses the problem in detail.

     Abel (2012) suggests that an approach to coinduction taken in MiniAgda avoids the problem. A possible alternative is to use copatterns (Abel et al. 2013; Abel and Pientka 2013). However, my impression is that the use of copatterns can make it harder to produce efficient code, so currently I am somewhat skeptical towards including them in the core of the language.

   - *Bisimilarity as the substitutive equality for coinductive types.*

     MiniAgda uses *intensional equality*, which is distinct from the more natural *extensional equality* that is typically used in mathematics. With intensional equality one can have two functions f and g that are not provably equal despite satisfying $f\ x = g\ x$ for all x. Similarly one can have two infinite lists that are not provably equal despite elements at corresponding positions being equal.

     The goal is to come up with a design that uses extensional equality, at least for coinductive types (where extensional equality amounts to bisimilarity). A decade ago this would have been a very ambitious goal. However, this changed with the advent of Observational Type Theory (Altenkirch et al. 2007), which uses extensional equality for functions; McBride (2009) later extended the design to coinductive types.

Some people at my department are currently trying to develop a computational foundation for "univalent foundations" (Voevodsky 2010), a flavour of type theory that uses extensional equality for all types. To avoid wasted work I plan to defer work on extensional equality until this work on a computational foundation has matured, and only then decide if the language design should be based on Observational Type Theory or univalent foundations.

If major complications should arise, then I plan to prioritise the other sub-tasks over this one.

2. *Turn this design into a practical, useful system.*

   I am one of the main developers of Agda, so it is natural to use Agda as the basis for this system. Work on Agda can also benefit other users of the language, and Agda is fairly popular at the moment:

   - Between March 2010 and February 2011 Agda was downloaded 2211 times from one of the sites which host it. (This is the latest figure that I have access to.)
   - More than 75 papers have used Agda since the current incarnation of the language was released in 2007, see the Agda Wiki (`http://wiki.portal.chalmers.se/agda/`).
   - The last Agda Implementors' Meetings (one-week meetings with talks, discussions and programming, held roughly twice every year, open not only to implementors but also to users of Agda) have had about 15–30 participants.
   - Agda is or has been taught at 12 or more universities (see the Agda Wiki).

   Work on this task can start once the design from the previous task starts becoming concrete. The task involves two steps:

   - Implementation of prototypes.
   - Changing Agda.

   The prototypes are used to experiment with implementation techniques and give us the opportunity to quickly evaluate design decisions. Feedback from the prototypes is likely to influence the language design.

   I expect that this task will be performed mainly by the PhD student, whose skills should mature as the project progresses, but I would still be involved.

3. *Develop new applications of nested induction and coinduction.*

   Such applications can be interesting in their own right, and can also demonstrate that the system that is developed is practical, and point to further improvements.

   Nested induction and coinduction is not a widely known technique, and I find it easy to discover applications of it, as witnessed by a series of papers (Danielsson and Altenkirch 2010; Danielsson 2010a; Danielsson and Norell 2011; Danielsson 2012b). Recently I have started a discussion with members of the security group at my department regarding the possibility to use nested induction and coinduction in work on language-based security.

The plan is that this task will be carried out in parallel with the tasks above, by both me and the PhD student.

4 Significance

In the field of programming languages it is nowadays quite common to see papers that are accompanied by machine-checked correctness proofs. If the project is successful, then researchers in this and other fields will have access to a relatively widely used tool with good support for nested (and non-nested) induction and coinduction. Furthermore the basic principles underlying the implementation could be reused in other tools like Coq.

More speculatively, the presence of an improved tool may also encourage more researchers to make use of nested induction and coinduction in their work. My personal experience is that a tool such as Agda makes this rather abstract concept more concrete and tangible, because you are not only working with mathematical definitions, but also with programs that you can run and values that you can inspect.

5 Preliminary results

As mentioned above I have already done work in this area, mostly related to the *use* of nested induction and coinduction (Danielsson 2010a,b, 2012b; Danielsson and Altenkirch 2010; Danielsson and Norell 2011). Through this work I have experienced some of the problems of the current approaches (Danielsson 2010b, 2012b; Altenkirch and Danielsson 2010). I was also involved in the design and implementation of the current, preliminary support for coinduction in Agda, which is partly based on work on the experimental language ΠΣ (Altenkirch et al. 2010). I think that my background, with a focus on the use of nested induction and coinduction but also experience of language design and implementation, puts me in a good position to develop a language design that is practically useful.

6 International and national collaboration

I collaborate with a number of persons from other universities:

- I am one of the key players in the Agda project. We have Agda Implementors' Meetings roughly twice every year. I co-organised two of the last five meetings; the last one I organised had about 30 participants from North America, Europe and Asia.

- I am collaborating closely with Andreas Abel (Ludwig-Maximilians-Universität München) who, together with Ulf Norell (Chalmers) and me, is the most prolific contributor to the Agda system. He also develops MiniAgda, with experimental support for sized types (see Sections 2–3).

  Andreas Abel has recently been offered a faculty position at my department. He has expressed interest in the proposed project, and I expect that he can make valuable contributions to it, whether he accepts the job offer or not.

- When I was a post-doc at the University of Nottingham I worked together with Thorsten Altenkirch on nested induction and coinduction.
- I have also had a number of discussions about coinduction and related issues with Lars Birkedal (Aarhus University), Conor McBride (University of Strathclyde), Anton Setzer (Swansea University), and Tarmo Uustalu (Tallinn University of Technology).

  Tarmo Uustalu is the manager of a project called "Coinduction for semantics, analysis and verification of communicating and concurrent reactive software" that was recently awarded research funding. I was included in the grant application as a member of the project's research group (but I get no money and have no formal obligations to do anything).
- I am currently an observer (a potential future member) of the IFIP Working Group 2.1 on Algorithmic Languages and Calculi.

7 Other grants

Last year I was listed as a coworker on a successful Swedish Research Council framework grant application (the grant holder is Thierry Coquand). The framework grant concerns a project that is different from, but related to, the one described here.

One of the other project's goals is to create a better Agda compiler, a safer foreign function interface, and libraries for writing safe, effectful code. Such libraries are likely to make use of coinduction, and can thus benefit from the work proposed in the present application.

Another goal is to create a computational foundation for "univalent foundations" (see Section 3). If this (difficult) line of work is successful, then we plan to make use of the new foundation in an updated version of Agda. As mentioned in Section 3 the proposed project does *not* depend on the success of the work on foundations—it is possible to base the design on Observational Type Theory instead.

8 Independent line of research

I started work on nested induction and coinduction as a post-doc, but I did not do this work together with my post-doc advisor, Graham Hutton. Three of my publications on this topic were written solely by me (Danielsson 2010a,b, 2012b), and one was based mostly on my own research (Danielsson and Altenkirch 2010). Another publication was a collaborative effort, based on previous, unpublished work by Altenkirch and Oury (Altenkirch et al. 2010), and for another publication I did the work related to nested induction and coinduction (Danielsson and Norell 2011). I have also recently submitted a paper that uses non-nested coinduction (Danielsson 2013).

Currently I am working on a project headed by Thierry Coquand. We investigate the nature of equality in certain variants of type theory. I have so far produced two papers directly related to this topic, one with a single author (Danielsson 2012a) and a recently submitted one with two authors (Coquand and Danielsson 2013).

I am not currently allowed to be the main supervisor of a PhD student, but I plan to apply for the title of "oavlönad docent", granting me the right to be a main supervisor, soon. At the moment I am the co-supervisor of one PhD student, Simon Huber.

References

Andreas Abel. Mixed inductive/coinductive types and strong normalization. In *APLAS 2007*, 2009. doi:10.1007/978-3-540-76637-7_19.

Andreas Abel. MiniAgda: Integrating sized and dependent types. In *PAR 2010*, 2010. doi:10.4204/EPTCS.43.2.

Andreas Abel. Type-based termination, inflationary fixed-points, and mixed inductive-coinductive types. In *FICS 2012*, 2012. doi:10.4204/EPTCS.77.1.

Andreas Abel and Brigitte Pientka. Wellfounded recursion with copatterns: A unified approach to termination and productivity. Submitted, available from `http://www2.tcs.ifi.lmu.de/~abel/`, 2013.

Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. Copatterns: Programming infinite structures by observations. In *POPL '13*, 2013. doi:10.1145/2429069.2429075.

The Agda Team. The Agda Wiki. Available at `http://wiki.portal.chalmers.se/agda/`, 2013.

Thorsten Altenkirch and Nils Anders Danielsson. Termination checking in the presence of nested inductive and coinductive types. Short note supporting a talk given at PAR 2010, available from `http://www.cse.chalmers.se/~nad/`, 2010.

Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. Observational equality, now! In *PLPV'07*, 2007. doi:10.1145/1292597.1292608.

Thorsten Altenkirch, Nils Anders Danielsson, Andres Löh, and Nicolas Oury. ΠΣ: Dependent types without the sugar. In *FLOPS 2010*, 2010. doi:10.1007/978-3-642-12251-4_5.

G. Barthe, M. J. Frade, E. Giménez, L. Pinto, and T. Uustalu. Type-based termination of recursive definitions. *Mathematical Structures in Computer Science*, 14(1):97–141, 2004. doi:10.1017/S0960129503004122.

Ulrich Berger. From coinductive proofs to exact real arithmetic: theory and applications. *LMCS*, 7(1:8), 2011. doi:10.2168/LMCS-7(1:8)2011.

Marc Bezem, Keiko Nakata, and Tarmo Uustalu. On streams that are finitely red. *LMCS*, 8(4:4), 2012. Available from `http://www.lmcs-online.org/ojs/viewarticle.php?id=971`.

Julian Bradfield and Colin Stirling. Modal mu-calculi. In *Handbook of Modal Logic*. Elsevier, 2007.

Michael Brandt and Fritz Henglein. Coinductive axiomatization of recursive type equality and subtyping. *Fundamenta Informaticae*, 33(4):309–338, 1998. doi:10.3233/FI-1998-33401.

The Coq Development Team. *The Coq Proof Assistant, Reference Manual, Version 8.4*, 2012. Available from `http://coq.inria.fr/`.

Thierry Coquand. Infinite objects in type theory. In *TYPES '93*, 1994. doi:10.1007/3-540-58085-9_72.

Thierry Coquand and Nils Anders Danielsson. Isomorphism is equality. Submitted, available from `http://www.cse.chalmers.se/~nad/`, 2013.

Nils Anders Danielsson. Total parser combinators. In *ICFP'10*, 2010a. doi:10.1145/1863543.1863585.

Nils Anders Danielsson. Beating the productivity checker using embedded languages. In *PAR 2010*, 2010b. doi:10.4204/EPTCS.43.3.

Nils Anders Danielsson. Bag equivalence via a proof-relevant membership relation. In *ITP 2012*, 2012a. doi:10.1007/978-3-642-32347-8_11.

Nils Anders Danielsson. Operational semantics using the partiality monad. In *ICFP'12*, 2012b. doi:10.1145/2364527.2364546.

Nils Anders Danielsson. Correct-by-construction pretty-printing. Submitted to ICFP'13, available from `http://www.cse.chalmers.se/~nad/`, 2013.

Nils Anders Danielsson and Thorsten Altenkirch. Subtyping, declaratively: An exercise in mixed induction and coinduction. In *MPC 2010*, 2010. doi:10.1007/978-3-642-13321-3_8.

Nils Anders Danielsson and Ulf Norell. Parsing mixfix operators. In *IFL 2008*, 2011. doi:10.1007/978-3-642-24452-0_5.

Eduardo Giménez. *Un Calcul de Constructions Infinies et son Application à la Vérification de Systèmes Communicants*. PhD thesis, Ecole Normale Supérieure de Lyon, 1996. Available at `ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/PhD/PhD96-11.ps.Z`.

Peter Hancock, Dirk Pattinson, and Neil Ghani. Representations of stream processors using nested fixed points. *LMCS*, 5(3:9), 2009. doi:10.2168/LMCS-5(3:9)2009.

John Hughes, Lars Pareto, and Amr Sabry. Proving the correctness of reactive systems using sized types. In *POPL '96*, 1996. doi:10.1145/237721.240882.

Hyeonseung Im, Keiko Nakata, and Sungwoo Park. Contractive signatures with recursive types, type parameters, and abstract types. Submitted, available from `https://www.lri.fr/~im/`, 2013.

Alan Jeffrey and Julian Rathke. The lax braided structure of streaming I/O. In *CSL'11*, 2011. doi:10.4230/LIPIcs.CSL.2011.292.

Vladimir Komendantsky. Subtyping by folding an inductive relation into a coinductive one. In *TFP 2011*, 2012. doi:10.1007/978-3-642-32037-8_2.

Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7): 107–115, 2009. doi:10.1145/1538788.1538814.

Paul Blain Levy. Infinitary Howe's method. In *CMCS 2006*, 2006. doi:10.1016/j.entcs.2006.06.006.

Conor McBride. Let's see how things unfold: Reconciling the infinite with the intensional. In *CALCO 2009*, 2009. doi:10.1007/978-3-642-03741-2_9.

Keiko Nakata and Tarmo Uustalu. Resumptions, weak bisimilarity and big-step semantics for While with interactive I/O: An exercise in mixed induction-coinduction. In *SOS 2010*, 2010. doi:10.4204/EPTCS.32.5.

Keiko Nakata, Tarmo Uustalu, and Marc Bezem. A proof pearl with the fan theorem and bar induction: Walking through infinite trees with mixed induction and coinduction. In *APLAS 2011*, 2011. doi:10.1007/978-3-642-25318-8_26.

Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers University of Technology and Göteborg University, 2007. Available from `http://www.cse.chalmers.se/~ulfn/`.

David Park. On the semantics of fair parallelism. In *Abstract Software Specifications, 1979 Copenhagen Winter School*, 1980. doi:10.1007/3-540-10007-5_47.

Christophe Raffalli. *L'Arithmétique Fonctionnelle du Second Ordre avec Points Fixes*. PhD thesis, Université Paris VII, 1994. Available from `http://www.lama.univ-savoie.fr/~raffalli/`.

Davide Sangiorgi and Robin Milner. The problem of "weak bisimulation up to". In *CONCUR '92*, 1992. doi:10.1007/BFb0084781.

Anton Setzer. Coalgebras in dependent type theory – the saga continues. Talk given at Agda Implementors' Meeting XII, available from `http://www.cs.swan.ac.uk/~csetzer/`, 2010.

Dmitriy Traytel, Andrei Popescu, and Jasmin Christian Blanchette. Foundational, compositional (co)datatypes for higher-order logic: Category theory applied to theorem proving. In *LICS 2012*, 2012. doi:10.1109/LICS.2012.75.

Tarmo Uustalu. Coinductive big-step semantics for concurrency. In *PLACES 2013, Preliminary Proceedings*, 2013. Available from `http://places13.di.fc.ul.pt/`.

Vladimir Voevodsky. Univalent foundations project (a modified version of an NSF grant application). Unpublished, available from `http://www.math.ias.edu/~vladimir/Site3/Univalent_Foundations.html`, 2010.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2013-40060-105052-38
Name of applicant
Danielsson, Nils Anders
Date of birth
791006-6013

Title of research programme
Practical nested induction and coinduction

Appendix B

Curriculum vitae

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

Appendix B: CV

Nils Anders Danielsson

1 Higher education degrees

2002 MSc in Engineering Physics, Chalmers University of Technology, Gothenburg. Obtained highest possible grade in every Chalmers course I took, received the John Ericsson medal as one of the best recent graduates from Chalmers.

2002 MSc in Advanced Computing, Imperial College of Science, Technology and Medicine, London. Thesis passed with distinction, A+ grade average (90%). (Taken as part of studies at Chalmers, so I was not formally awarded this degree.)

2 Doctoral degree

2007 PhD in Computing Science, Chalmers University of Technology, Gothenburg. Title of thesis: Functional Program Correctness Through Types. Supervisor: Patrik Jansson.

3 Postdoctoral positions

2008–2010 Research Fellow, University of Nottingham.

5 Present position, period of appointment, percentage of research in the position

2011–2014 Assistant professor, University of Gothenburg. 80% research.

9 Other information of importance to the application

Peer Review

PC member POPL 2013, DTP 2013, Haskell 2012, LFMTP 2012, MSFP 2012, TFP 2010.

Co-editor TYPES 2011 (post-proceedings).

Reviewer POPL, ICFP, LICS, ESOP, TPHOLs, ITP, CSL, JFP, JLC, LMCS, Haskell, PEPM, PPDP, MSFP, TFP, TLDI, PLPV, WoLLIC, Acta Informatica, Journal of Logic and Algebraic Programming.

Other Community and University Services

2011–2013 Seminar organiser, Gothenburg.

2010–2011 Organiser of the twelth and thirteenth Agda Implementors' Meetings, Nottingham and Gothenburg.

2010 Seminar organiser, Nottingham.

2007 Member of the organising committee for the sixth Agda Implementors' Meeting, Gothenburg.

2003–2006 Member of the programme committee for the MSc programme in Software Engineering at Chalmers.

2003 Member of the organising committee for the ICFP Programming Contest.

2001–2002 Year representative for the students of the Master of Advanced Computing programme at Imperial College.

2000–2001 Course evaluator for the Engineering Physics programme at Chalmers.

Awards, Grants, etc.

2004 + 2007 Recipient of grants from Stiftelsen Claes Adelskölds medalj- och minnesfond (twice) and Chalmersska forskningsfonden (also twice), used for research visits and conference travel.

2004 Recipient of the John Ericsson medal as one of the best recent graduates from Chalmers.

1999–2002 Recipient of awards from Anna Whitlocks Minnesfond, Chalmers donationsstipendier, Odd Alberts donationsfond, Adlerbertska Stipendiefonden and Telefondirektören H.T. Cedergrens Uppfostringsfond, some of them for excellence in studies.

1998 Top 20 in the Swedish and Nordic secondary school mathematics contests (7th and 15th) and the Swedish secondary school physics contest (17th).

1995 4th in the Swedish primary school mathematics contest.

Appendix C

List of publications

Appendix C: Publication list

Nils Anders Danielsson

Quick summary: POPL (twice), ICFP (twice), FLOPS, ITP, MPC, TYPES, IFL, PAR. Note that all but two of the papers below (FLOPS 2010 and POPL 2006) are accompanied by machine-checked proofs, available from `http://www.cse.chalmers.se/~nad/`. Citation data is available on my Google Scholar page (`http://scholar.google.com/citations?user=YMU90ywAAAAJ`).

2 Peer-reviewed conference contributions

* Nils Anders Danielsson. Operational semantics using the partiality monad. In *ICFP'12, Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming*, pages 127–138, 2012. doi:10.1145/2364527.2364546.

Nils Anders Danielsson. Bag equivalence via a proof-relevant membership relation. In *Interactive Theorem Proving, Third International Conference, ITP 2012*, volume 7406 of *LNCS*, pages 149–165, 2012. doi:10.1007/978-3-642-32347-8_11.

* Nils Anders Danielsson. Total parser combinators. In *ICFP'10, Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*, pages 285–296, 2010. doi:10.1145/1863543.1863585.

* Nils Anders Danielsson. Beating the productivity checker using embedded languages. In *Proceedings Workshop on Partiality and Recursion in Interactive Theorem Provers (PAR 2010)*, volume 43 of *EPTCS*, 2010. doi:10.4204/EPTCS.43.3.

* Nils Anders Danielsson and Thorsten Altenkirch. Subtyping, declaratively: An exercise in mixed induction and coinduction. In *Mathematics of Program Construction, 10th International Conference, MPC 2010*, volume 6120 of *LNCS*, pages 100–118, 2010. doi:10.1007/978-3-642-13321-3_8.

* Thorsten Altenkirch, Nils Anders Danielsson, Andres Löh, and Nicolas Oury. ΠΣ: Dependent types without the sugar. In *Functional and Logic Programming, 10th International Symposium, FLOPS 2010*, volume 6009 of *LNCS*, pages 40–55, 2010. doi:10.1007/978-3-642-12251-4_5.

Nils Anders Danielsson and Ulf Norell. Parsing mixfix operators. In *Implementation and Application of Functional Languages, 20th International Symposium, IFL 2008*, volume 5836 of *LNCS*, pages 80–99, 2011. doi:10.1007/978-3-642-24452-0_5.

Nils Anders Danielsson. Lightweight semiformal time complexity analysis for purely functional data structures. In *POPL'08: Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 133–144, 2008. doi:10.1145/1328438.1328457.

Nils Anders Danielsson. A formalisation of a dependently typed language as an inductive-recursive family. In *Types for Proofs and Programs, International Workshop, TYPES 2006, Revised Selected Papers*, volume 4502 of *LNCS*, pages 93–109, 2007. doi:10.1007/978-3-540-74464-1_7.

Nils Anders Danielsson, Jeremy Gibbons, John Hughes, and Patrik Jansson. Fast and loose reasoning is morally correct. In *Conference record of POPL 2006: The 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 206–217, 2006. doi:10.1145/1111037.1111056.

5 Open-access computer programs that you have developed

Agda I am one of the three main developers of *Agda*, a dependently typed functional programming language.

Available from `http://wiki.portal.chalmers.se/agda/`.

Agda's standard library Agda's standard library is developed principally by me.

Available from `http://wiki.portal.chalmers.se/agda/`.

Total Parser Combinators A parser combinator library for Agda (see the paper "Total parser combinators").

Available from `http://www.cse.chalmers.se/~nad/software.html`.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod

þÿ 2 0 1 3 - 4 0 0 6 0 - 1 0 0 2 -

Name of applicant

Danielsson, Nils Anders

Date of birth

791006-6013

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

Appendix N: Budget and research resources

Nils Anders Danielsson

1 Justification of the budget

My salary I plan to spend 40% of my time on this project, and apply for 40% of my salary costs for 2015–2018. (I do not apply for salary costs for myself for 2014.)

The PhD student's salary For the PhD student I apply for 80% of the salary costs for the full five years of her or his education (the plan is to fund the remaining 20% via the teaching budget).

Computer equipment and literature Computers and related equipment; literature.

Travel and conferences Conferences and workshops; research meetings and visits; summer schools (including schools taking place in other seasons); travel costs for licentiate discussion (discussion leader) and PhD defence (opponent and grading committee).

Publication costs Printing of licentiate and PhD theses.

2 Total research resources of the project

Type of grant	Applied or granted	Funding source	Grant holder/ Project leader	Grant period	Total amount (kSEK)
ERC Advanced Grant	Granted	ERC	Thierry Coquand	2010-04–2015-03	$\sim$ 16 000 (1 922 kEUR)
Framework Grant	Granted	SRC	Thierry Coquand	2013–2016	(12 000)
Project Grant Junior Researchers	Applied	SRC	Nils Anders Danielsson	2014–2018	5 681

I expect that, until the end of 2014, a large part of my salary will be paid for with funds from the ERC grant listed above, so I only apply for salary costs for myself for 2015–2018. For all other expenses I apply for the full budgeted cost.

The SRC framework grant listed above may be used to pay for some of my costs (we applied for funding for an estimated 20% of my salary costs, but did not get all the money that we applied for). However, this grant concerns a different project than the one I am applying for—see Appendix A—so I have put the amount in parentheses. Note that I only plan to spend 40% of my time on the project proposed in the present application, so some of the remaining time can be spent on the other project.

The following table shows the proportion of the project's budgeted costs that I apply for, year by year:

Year	Proportion (%)
2014	60
2015	100
2016	100
2017	100
2018	100

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2013-40060-105052-38

Dnr
2013 -

Name of applicant
Danielsson, Nils Anders

Date of birth
791006-6013

Reg date
2013-04-11 19:00:40

Project title
Practical nested induction and coinduction

NT

NT-2, NT-1

Research grants NT April 11, 2013
Project Grant Junior Researchers
*Nat-Tek generellt
Göteborgs universitet

Appendix S

Signatures

2014	2015	2016	2017	2018
698	1108	1222	1254	1399

2014-01-01 -- 2018-12-31

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

2013-40060-105052-38

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