VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kansliets noteringar Kod	Dnr
2014-42285-115143-47	2014-4849

2014
Project Grant Junior Researchers

Area of science
Natural and Engineering Sciences
Announced grants
Research grants NT April 9, 2014
Total amount for which applied (kSEK)

2015	2016	2017	2018	2019
736	967	1000	1031	258

APPLICANT

Name(Last name, First name)
Johansson, Moa

Date of birth
810720-8962

Gender
Female

Email address
moa.johansson@chalmers.se

Academic title
PhD

Position
Post doc

Phone
031 7721078

Doctoral degree awarded (yyyy-mm-dd)
2009-11-29

WORKING ADDRESS

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

41296 Göteborg,

ADMINISTRATING ORGANISATION

Administrating Organisation
Chalmers tekniska högskola

DESCRIPTIVE DATA

Project title, Swedish (max 200 char)
Lärande och teoribildning inom automatisk teorembevisning

Project title, English (max 200 char)
Learning and Exploration in Automated Theorem Proving

Abstract (max 1500 char)
Software bugs can have serious consequences as computers are used more widely in all parts of society. Formal verification techniques can provide a higher level of assurance than just testing, by mathematically proving that the program satisfies certain properties. However, these proofs often require considerable time and expertise. Development of automated techniques for verification is thus of importance.

Different verification tasks requires different proof techniques. This project concerns automation of inductive proofs, needed to prove properties about recursive structures and functions. One of the main challenges in automated inductive theorem proving is to find missing lemmas. Our approach to lemma discovery integrates automated theorem proving, machine learning and theory exploration. Theory exploration techniques can discover basic properties and lemmas in new theory developments. We want to also exploit knowledge available in existing large proof libraries, using machine learning. We will extend theory exploration techniques to also use information learned from previous proofs to extract heuristic guidance in the search for new lemmas and proofs.

This project will make contributions towards scalable inductive theorem proving and theory exploration, which is crucial for handling proofs arising from program verification. We plan to apply our results to reasoning about programs written in the functional programming language Haskell.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2014-42285-115143-47

Name of Applicant
Johansson, Moa

Date of birth
810720-8962

Abstract language
English

Keywords
theorem proving, machine learning, induction, functional programming, theory exploration

Review panel
NT-2

Project also includes other research area

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

Aspects

Continuation grant
Application concerns: New grant
Registration Number:

Application is also submitted to

similar to: **identical to:**

ANIMAL STUDIES

Animal studies
No animal experiments

ENCLOSED APPENDICES

A, B, C, N, S

APPLIED FUNDING: THIS APPLICATION

Funding period (planned start and end date)
2015-04-01 -- 2019-03-31

Staff/ salaries (kSEK)

Main applicant	% of full time in the project	2015	2016	2017	2018	2019
Moa Johansson	80	637	877	907	937	242
Other staff						
	Total, salaries (kSEK):	637	877	907	937	242

Other project related costs (kSEK)	2015	2016	2017	2018	2019
Conference and Travel	30	30	30	30	
Computer purchase	25				
Premises	35	48	50	51	13
IT costs	9	12	13	13	3
Total, other costs (kSEK):	99	90	93	94	16

Total amount for which applied (kSEK)

2015	2016	2017	2018	2019
736	967	1000	1031	258

ALL FUNDING

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

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2014-42285-115143-47

Name of Applicant
Johansson, Moa

Date of birth
810720-8962

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

POPULAR SCIENCE DESCRIPTION

Popularscience heading and description (max 4500 char)

Lärande och Teoribildning inom Automatisk Teorembevisning

Fel i datorprogram, så kallade buggar, kan få stora konsekvenser. År 2002 beräknades buggar i program kosta den amerikanska ekonomin $60 miljarder per år enligt en undersökning av National Institute for Standards and Technology. Den siffran är knappast lägre idag, när datorer används i än större utsträckning till en mängd vitt skilda uppgifter. Kostnaden för att fixa en bug kan vara ännu högre om den upptäcks efter att datorsystemet har börjat användas. Toyota tvingades till exempel år 2010 återkalla 400 000 Prius-modeller efter att en bug hade upptäckts i programmet som kontrollerade bromssystemet.

Att enbart testa program kan inte ge hundraprocentiga garantier för att det inte finns några buggar eftersom det är praktiskt omöjligt att testa alla möjliga kombinationer av indata till stora system. Formella verifikationsmetoder har därför vuxit i popularitet och blivit allt viktigare som komplement till testning.

Formella metoder används för att matematiskt bevisa att programmet verkligen har vissa önskvärda egenskaper, och t.ex. därför aldrig kommer att krascha på vissa sätt. Tyvärr är sådana bevis ofta komplicerade och kräver både tid och expertis, något som de flesta programmerare saknar. Viss hjälp kan fås av program som kallas teorembevisare och kan assistera eller ibland t.o.m. helt automatiskt bevisa vissa egenskaper.

Många etablerade teorembevisare har numer stora elektroniska matematiska bibliotek som användaren kan bygga vidare på för att skapa nya matematiska teorier eller som hjälp med nya bevis. Tyvärr kan bibliotekens storlek och inkonsekvent representation stå i vägen för effektiv återanvändning av kunskapen som finns tillgänglig. Ofta kan det vara lättare att skapa ett nytt matematiskt teoribibliotek istället för att ägna tid åt att söka efter liknande teorier som kan anpassas till ett nytt sammanhang.

Vårt projekt har som syfte att bättre utnyttja dessa bibliotek för att öka automatiseringsgraden i teorembevisare och därigenom göra dem lättare att använda. Vi kommer att använda tekniker för maskininlärning för att bygga statistiska modeller baserat på data i bevisbibliotek, för att t.ex. kunna identifiera likheter mellan olika bevis. Statistiska metoder som maskininlärning är bra på att hitta mönster i stora mängder data men kan inte förklara varför dessa mönster uppstått. Vi vill därför kombinera detta med en ny teknik som vi kallar Teoribildning (Theory Exploration). Ett teoribildningssystem har som uppgift att automatiskt skapa en lämplig matematisk representation, eller teori, av t.ex. ett datorprogram som vi vill verifiera. Desto rikare matematisk teori som vi kan ge till en teorembevisare, desto kraftfullare blir den, och fler bevis kan då klaras av helt automatiskt. För att ett teoribildningssystem ska vara effektivt kan information från maskininlärning vara till stor hjälp. Om vi exempelvis försöker bevisa någon ny egenskap hos vårt program kan teoribildningssystemet först få information om liknande bevis från maskininlärningssystemet, och sedan använda dessa liknande bevis för att söka efter ett analogt bevis för den nya egenskapen på ett effektivt sätt.

Olika sorters program och egenskaper behöver olika bevistekniker. Vi kommer att fokusera på induktionsbevis, som behövs för att bevisa egenskaper hos program som är baserade på rekursion. Sådana program räknar ut en lösning på ett problem genom att kombinera lösningar på mindre instanser av samma problem. Funktionella programmeringsspråk, till exempel språket Haskell, är exempel där rekursion förekommer mycket ofta. För att bevisa egenskaper hos funktionella program behövs i regel induktionsbevis, men i befintliga verifikationssystem måste induktionsbevis oftast göras interaktivt, med den mänskliga användaren som guidar bevisets olika steg. För att kunna göra det krävs, som tidigare nämnts, en hel del kunskap, tid och erfarenhet av liknande bevis. Vårt mål är att utveckla nya tekniker för att automatisera induktionsbevis som kan integreras i verktyg för verifiering av program. En programmerare kommer till exempel att kunna använda våra tekniker till att automatiskt kontrollera och bevisa att programmet fungerar som avsett.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2014-42285-115143-47

Name of applicant
Johansson, Moa

Date of birth
810720-8962

Title of research programme
Learning and Exploration in Automated Theorem Proving

Appendix A

Research programme

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

APPENDIX A: LEARNING AND EXPLORATION IN AUTOMATED THEOREM PROVING

MOA JOHANSSON

1. Purpose and Aims

Software systems are today prevalent throughout our daily lives, for instance in networking, security, transport (cars, aeroplanes, trains) and medical devices. Failure of such systems may cause dangerous situations or have huge financial implications. For instance, the U.S. National Institute of Standards and Technology estimates that software faults costs the U.S. economy nearly $60 billion dollars per year [30]. Software testing alone cannot guarantee correctness, it is infeasible to test all possible combinations of inputs of a large program. Formal verification techniques are an increasingly important complement, providing a higher level of assurance by mathematically *proving* that the program satisfy certain criteria. Several large companies such as Microsoft, IBM and Intel have started developing and using formal methods tools. However, a key limiting factor for wider commercial uptake is that formal proofs are difficult and time consuming, thus increasing costs in terms of development time and necessary expertise.

Depending on the kind of properties and the structure of the program we wish to verify, different techniques are required. Among the main automated techniques often used for verification are satisfiability modulo theories (SMT), model checking and first-order theorem proving. However, none of these techniques can handle proofs by induction which are needed if we want to prove properties about, for example, programs with recursion in functions or data-structures, or programs with loops. While inductive properties are common, inductive proofs are hard to automate, in particular because it is difficult to discover necessary intermediate results (lemmas), which may themselves require another inductive proof. Inductive proofs are therefore often performed interactively, using software *proof assistants*, where the user guides the search for a proof. This does however require domain knowledge and expertise from the user. An underexploited resource in automated theorem proving is the already existing electronic libraries of previously proved mathematics for various theorem proving systems. We propose a project for utilising this knowledge for *automation of inductive proofs*.

The purpose of this project is to combine the advantages of statistical machine learning methods with those of symbolic methods for lemma discovery, such as theory exploration. We can thus better exploit the resources in large electronic proof libraries to gain heuristic knowledge and build more efficient lemma discovery techniques. This will result in scalable, more powerful methods for automated inductive theorem proving which will be applied to problems from program verification.

The size as well as technical and notational sophistication of proof libraries often stand in the way of efficient knowledge re-use. It is often easier to start a new library from scratch rather than searching existing ones for lemmas that can be re-used or adapted to a new context. In our previous work on *theory exploration*, we have developed techniques which can discover basic properties and lemmas in new theory developments. In this project, we go beyond these techniques by also utilising information from previous proofs to extract heuristic guidance in the search for new lemmas and proofs.

Machine learning and *pattern recognition* techniques are statistical methods which can be used for discovering similarities in data. Statistical methods are well suited for fast processing of big libraries and are generally tolerant to noise. While such statistical methods focus on extracting information from large volumes of data, they lack in capacity for conceptualisation. They can, for example, tell that a correlation between two proof patterns exists, but not why this happens nor formulate any conceptual proof hints for similar situations. Theory exploration

techniques on the other hand, are concerned with exactly that: the discovery of new lemmas in new theories. However, theory exploration techniques may not cope well with very large theories when the search space becomes too big. We thus want to combine these two techniques exploiting the strengths of both. For example, suppose we try to prove a new conjecture. We will use pattern recognition and data mining to detect if some similar theorem has been proved before, and if so, extract information about which proof technique and which lemmas were used. This information can then be fed into a theory exploration system to restrict its search space and produce suggestion of, for example, analogous lemmas applicable to the new conjecture.

Within the scope of this project we will evaluate our techniques primarily on verification problems of functional programs written in the Haskell language. Functional programs are structured by recursion, why many properties require inductive proofs. The results are however applicable to inductive proofs in general, and can be used to reason about programs written in other languages as well. The proposed work will advance the state-of-the-art and is distinguished from existing techniques in the following main ways:

- Our techniques are a complement to testing. While tools such as QuickCheck [3], can find many bugs by randomly generated test-cases for user stated properties, we will provide a higher level of assurance by proving that such properties hold.
- We take a novel approach to automated induction by combining it with statistical machine learning and theory exploration techniques to create a richer background theory for reasoning. With a sufficiently rich background theory, a theorem prover can succeed in proving many difficult problems with a relatively simple strategy.
- We will improve on current lemma discovery techniques by exploiting information available in large electronic libraries. Our preliminary work shows that applying machine learning to these libraries can provide hints and heuristic guidance allowing for a reduction in the search space of theory exploration algorithms [12]. For instance, we can construct algorithms based on reasoning by analogy.
- Statistical methods are robust to noise, deal well with large volumes of data and are adaptable to new knowledge. As new definitions and proofs are constructed and added to the proof library, new patterns may emerge, suggesting even more precise hints. This makes the proposed work applicable in both automated and interactive theorem proving systems.

2. Survey of the Field

Theorem Proving for Program Verification. Satisfiability Modulo Theories (SMT) solving is a popular automated theorem proving technique for many verification problems. It works well for combinations of particular decidable theories and on large variable-free problems. First-order logic (FOL) theorem proving is another mature technique, which is strong for fast fully automated reasoning about problems that contains variables. However, within neither of these domains there has been much attention to addressing proofs by induction. Inductive proofs are required to reason about recursive data structures and functions. We strongly believe that many more inductive proofs could be automated than is currently possible. One challenge in inductive proofs is that we may need intermediate results, in the form of auxiliary lemmas or generalisations, which are difficult to find and prove automatically. Traditionally, these issues has been circumnavigated by performing inductive proofs interactively, for instance using proof assistants such as ACL2 [18] or Isabelle [26], which requires considerable expertise from the user.

Although the uptake of automated inductive theorem proving for verification purposes has been slow so far, there has been research in the area. *Proof-planning* is a theorem proving technique which exploits the fact that certain families of proofs share a similar structure [1]. For instance, all inductive proofs consists of one or several base-cases followed by step-cases. If the proof-plan does not succeed, *proof-critics* may be employed to attempt to analyse why the proof failed, and perhaps suggest a lemma [14]. The IsaPlanner system is a state-of-the-art

proof-planner, which performs fully automatic inductive proofs [10]. It is built on top of the popular interactive theorem prover Isabelle [26]. IsaPlanner can perform structural induction over datatypes and then proceeds by rewriting, guided by the rippling heuristic towards application of the inductive hypothesis [2]. IsaPlanner have critics for automated lemmas discovery, which works well for simpler cases when enough progress has been made so that the inductive hypothesis can be applied and the lemma deduced from the remaining subgoal. However, for more complicated proofs, the critics often fail to produce useful lemmas [15].

Recently, there has been renewed interest in inductive theorem proving for program verification. Zeno is a rewriting based theorem prover for the terminating and total subset of Haskell, also supporting simple lemma discovery like IsaPlanner [29]. Dafny is a programming language developed at Microsoft Research, with integrated support for stating properties which can be automatically proved by the SMT-solver Z3 [21]. Dafny can identify which properties that are likely to need induction, but relies on the user to provide necessary lemmas.

Machine Learning in Theorem Proving. Machine learning techniques has been used in various ways in the context of automated theorem proving. In first-order provers, the speed at which they find a proof is dependent on many things, including which subset of premises are to be passed to the prover if the problem at hand is very large. Too many, and the performance will degrade, too few, and the problem may become unsolvable. Machine learning techniques can help by using information from previous proofs to find correlations between theorem statements and the facts used in their proofs. This in turn can help guide premise selection when a similar conjecture is attempted. This has been implemented in for instance the provers E and SPASS [9, 20, 31].

The interactive proof assistant Isabelle supports calling external automated first-order provers, using the Sledgehammer tool [27]. When given a new conjecture, Sledgehammer must choose which of the many thousands of facts and lemmas present in Isabelle's libraries it will send off to the external provers, for which it uses a machine learning implementation of the naive Bayes algorithm [19].

The above works are mainly concerned with picking the right *existing* lemmas or facts out of a large library. We want to extend the scope to also *invent* such lemmas or facts if they are missing and the proof fails, using learned information about similar facts, proofs and lemmas as guidance.

Testing and Counter-Examples. QuickCheck is a very successful tool for testing Haskell programs automatically [3]. The programmer provides a specification of the program as a set of properties the program should satisfy. QuickCheck then automatically generates a large set of random test cases, to check these properties. Should some test case cause one of the properties to be false, this case can be returned to the user as a counter-example, providing a useful hint of where to look for a bug. Our work builds further upon this: In addition to providing counter-examples for false properties, we want to provide proofs for properties that are true. This way we get an even higher level of assurance than testing only, in particular we can be sure that there are no rare inputs, which had not been tested, that would cause an error. However, testing and counter-example finding still play an important role in the theory exploration techniques we use to generate candidate background lemmas.

Theory Exploration. Theory exploration is a recent technique, which aims at discovering lemmas about a given set of datatypes and functions, thus 'exploring' and building up a background theory. These systems typically generate candidate terms, which satisfy some 'interestingness' criteria, and then filter these through a counter-example checker. The space of all possible candidates is typically large, so theory exploration systems use various heuristics to narrow down the search space. Most theory exploration systems have been designed for applications in mathematics in mind, rather than program verification, such as the HR and MATHsAiD systems [7, 23]. However, HR was recently extend in the HREMO system, which use theory exploration to discover invariants for Event-B models [22].

Proof assistants, such as Isabelle [26] have large libraries of theorems and lemmas that users can build further upon. The theory exploration systems IsaCoSy, developed during the applicants PhD [16], and IsaScheme [24] have been developed for inductive theories in Isabelle. Both these systems have been shown able to automatically produce lemmas of the kind present in the libraries.

3. Preliminary Results

The proposed project builds on preliminary work described in two of our papers from last year, both presented at top conferences for automated theorem proving and deduction. The proposed project will consolidate these two strands of research.

The first article describes HipSpec [5], a state-of-the-art inductive theorem prover for Haskell developed at Chalmers. It was presented at the Conference on Automated Reasoning (CADE). HipSpec is the first theorem prover which fully integrates theory exploration to discover interesting lemmas which it adds to its background theory.

The second paper [12], was written together with colleagues in Dundee and Edinburgh and presented at the Conference on on Logic for Programming, Artificial Intelligence and Reasoning (LPAR). It describes a novel method for combining statistical machine learning with symbolic methods for lemma discovery, demonstrating the feasibility of the work proposed in this project.

HipSpec: Inductive Theorem Proving and Theory Exploration. HipSpec uses theory exploration techniques to automatically discover interesting lemmas which it adds to its background theory. HipSpec is the first theorem prover which fully integrates theory exploration in this way; it takes a Haskell program as input and produce a set of equational theorems about the functions and datatypes present. HipSpec has two modes of use:

Theorem Proving mode: HipSpec is used as an automated theorem prover of specific user defined properties. It first applies theory exploration to produce a background theory, and then attempt to prove the user-given conjecture using this richer theory.

Theory Exploration mode: HipSpec is used to automatically discover and prove properties about the program, using heuristics to select a small number of interesting properties which are presented to the user as a concise mathematical description of the program.

Our experimental results were very encouraging: On test-suites from the literature, HipSpec proved more problems than other state-of-the-art inductive provers such as Zeno, ACL2, IsaPlanner and Dafny [29, 18, 10, 21], including several problems which could not be proved by any of the other systems. Among the problems only proved by HipSpec was theorems posed as open challenge problems in 2005 ([2], p.77).

Hipster: An integration of HipSpec into Isabelle. Also interactive theorem provers strive to provide a lot of built in automation for users. We recently integrated our automated induction techniques from HipSpec with the widely used Isabelle proof assistant [17]. Hipster is an automated extension to Isabelle for inductive proofs using HipSpec as a backend. Like HipSpec, it can be used in theory exploration mode, to discover basic lemmas in a new theory development. These are then imported back into the Isabelle theory. In addition, Hipster can be called in the middle of an interactive proof attempt, when the human user is stuck and needs help to find an appropriate missing lemma.

Machine Learning and Lemma Discovery. Together with colleagues working in machine learning and theorem proving at the universities in Dundee and Edinburgh, we proposed a novel technique for combining statistical machine learning for proof-pattern recognition with symbolic methods for lemma discovery. A proof-of-concept tool, called **ACL2(ml)**, was implemented for assessing the feasibility of the approach [12]. The aim was to show that by using information gathered about previous proof attempts, we could improve the scalability and speed of lemma discovery algorithms. We studied some examples where HipSpec performs badly due to a large search space and found that the search space of the lemma discovery process indeed could be reduced, by identifying analogous proofs.

4. Project Description

We will advance the state-of-the-art in automated inductive theorem proving and theory exploration by developing new algorithms that take as heuristic guidance the insights provided by statistical machine learning methods. We will apply our methods to reason about Haskell programs, as well as integrating them with widely used theorem proving system such as Isabelle. Below, we outline the different milestones and work packages in the project:

WP1: Evaluation and experimentation with feature extraction and learning algorithms suitable for proof recognition.

WP2: Development of theory exploration algorithms exploiting machine learning. We mention three applications:

- **2A:** Discovering lemmas by analogy from similar proofs.
- **2B:** Speculating new properties by analogy to properties of similar functions.
- **2C:** Bridging theories between similar datatypes.

WP3: Implementation and Tools. We will integrate the techniques with the Hipster and HipSpec systems and start building up a corpus of verified Haskell code.

WP4: Evaluation and development of typical use cases.

In Figure 1 we give a high-level overview of the proposed system. The machine learning system will read information about proofs from the existing theory libraries, extracting hints which will be passed to the theory exploration sub-system. This in turn, is responsible for speculating a set of interesting new conjectures in accordance with requirements. Conjectures are then passed to the automated theorem prover, after which those proved gets added to our new theory and becomes available for future proofs.

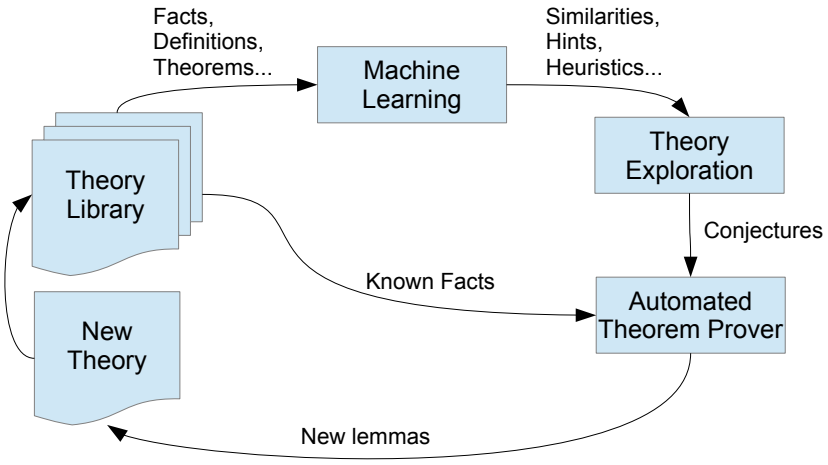

Figure 1. Workflow of the proposed system

WP1: Learning for Theory Exploration. There are a wide range of algorithms from the machine learning domain for both feature extraction and learning from those features. As an initial step we plan to survey the options and compare to existing machine learning applications from the theorem proving domain. Typically, rather basic methods, such as naive Bayes have been used. We will conduct a more thorough survey of the different options. What features we should choose to learn from also needs to be determined by experimentation. For example, we may extract features from the term-tree structure of conjectures and theorems, from the recursive structure and number of arguments of functions and datatypes, from typing information and so on. Potentially, a MSc student could be recruited to help with this part of the project.

WP2: Theory Exploration Algorithms. We identify three initial directions where the output from statistical methods can inform a theory exploration algorithm. We expect to use methods based on clustering to group together, for instance, open conjectures with previously proved

theorems or different functions and datatypes which appear to share similar definitional structure. This information will then be fed to the theory exploration system.

WP2A: Learning from similar proofs. In this case, we expect to cluster a new conjecture statement with existing theorems and proofs, using features based on their term structure and the similarity of the functions appearing in each. The hypothesis is that lemmas used in the proof of a similar theorem can be used to extract an *analogous lemma* for our new conjecture.

Example: Suppose we have two implementations of the factorial function in our proof library, one primitive recursive and one more efficient, using an accumulator and tail recursion:

$$\begin{aligned} &fact(0) = 1 && fact\_tail(0,\ a) = a \\ &fact(Suc(x)) = x\ *\ Suc(x) && fact\_tail(Suc(x),\ a) = fact\_tail(x,\ (a * x)) \end{aligned}$$

A typical property we may have in a proof library is that the two implementations produce the same result: $fact(x) = fact\_tail(x,\ 1)$. To prove this by structural induction, we will need a lemma, namely the generalisation: $fact(x) * a = fact\_tail(x,\ a)$, of which the theorem is a special case when $a = 1$. Now suppose that we want to tackle a similar problem, proving equivalence between two implementations of the fibonacci function:

$$\begin{aligned} &\mathit{fib}(0) = 0 && \mathit{fib\_tail}(0,\ j,\ k) = j \\ &\mathit{fib}(1) = 1 && \mathit{fib\_tail}(1,\ j,\ k) = k \\ &\mathit{fib}(Suc\ (Suc(x))) = \mathit{fib}(Suc(x)) + \mathit{fib}(x) && \mathit{fib\_tail}(Suc(x)),\ j,\ k) = \mathit{fib\_tail}(x,\ k,\ (j + k)) \end{aligned}$$

To prove the theorem: $fib(x) = fib\_tail(x,\ 0,\ 1)$, we will again need an auxiliary lemma, just as in the case for factorial. But how do we find it?

As we demonstrated in [12], statistical machine learning can at this point be used to check which known theorems are similar in structure to our conjecture about fib and will suggest that the previous proof about the factorial function might contain hints as to how to solve this new problem. In particular, we can use that lemma to try to construct another one applicable to this new proof situation. The information from the statistical model can tell us that the functions $fact$ and fib are similar, as are $fact\_tail$ and fib_tail. As a first step of the lemma discovery algorithm, we use the lemma about $fact$ as a *template*, by simply replacing $fact$ by fib and adding extra variables as they take different numbers of arguments. Secondly, we will have to do some search and testing, as in HipSpec, exploring terms produced by further instantiating variables or adding extra term structure. However, the search space is greatly reduced compared to searching over all possible terms. Using such steps, as demonstrated in [12], it is possible to find the desired lemma: $(fib(x - 1) * j) + (fib(x) * k) = \mathit{fib\_tail}(x,\ j,\ k)$.
Notice that the original theorem we started off trying to prove again is a special case of the lemma, when $j = 0$ and $k = 1$.

WP2B: Learning from similar functions. Here, we expect to cluster similar function definitions together, with the hypothesis that if a function f is similar to a function g, and we know some properties about f, then it is worth exploring if g has some similar properties. Of course, this will not always be the case, but at least gives some guidance as to which properties *might* hold about a new function, thus reducing the search space.

Example: Consider the function rev which reverses a list and the function $mirror$ which flips a tree data-structure around. Existing theory exploration techniques can very quickly find properties about a single function, for example that both rev and $mirror$ are idempotent: $rev(rev(l)) = l$ and $mirror(mirror(t)) = t$. Now, these two functions appear to have similarities, both in their definitional structures, argument types and properties. Further suppose that we have a function map which applies a function to all items in a list and a function $tmap$ which does the same for trees:

$$map(f,\ []) = []$$
$$map(f,\ (x : xs)) = f(x) : map(f,\ xs)$$

$$tmap(f,\ Leaf) = Leaf$$
$$tmap(f,\ Node(left,\ data,\ right)) = Node(tmap(f,\ left),\ f(data),\ tmap(f,\ right))$$

The types of these functions are similar, taking a function and a list or tree as arguments. They are also similar in that they have no work to do for the empty list and Leaf-case respectively. Now, assuming we have properties proved about lists in our library, for instance how map distribute over rev: $map(f,\ rev(l)) = rev(map(f,\ l))$. We can then directly speculate that similar properties hold also about the corresponding functions on trees, $mirror$ and $tmap$, for example: $tmap(f,\ mirror(t)) = mirror(tmap(f,\ t))$. In this case, a quick direct replacement of functions is sufficient, but in general some restricted search may apply. However, we greatly restrict the search space compared to traditional theory exploration techniques.

WP2C: Learning from similar datatypes. Suppose that we have two theories about two datatypes and functions on these, and furthermore, that some properties have been proved over the two datatypes. If many of these properties are similar, it is worth speculating some *bridging functions* to connect the two theories, for instance converting one datatype into the other.

Example: Suppose that we have applied the above techniques and found several similar function pairs such as rev and $mirror$, map and $tmap$ over List and binary trees, as well as similar properties over these. Let us speculate a function which converts from trees to lists. One possibility is a function which simply traverses the tree, adding data items to a list from left to right:

$$toList(Leaf) = []$$
$$toList(Node(left,\ data,\ right)) = toList(left) +\!\!+ [data] +\!\!+ toList(right)$$

Of course, there will be other options as to the order in which to traverse the tree, so this algorithm will be allowed to speculate several such functions. We can limit the algorithm by heuristics on, for instance, recursive structure and complexity of the generated function body.

WP3: Implementation and Tools. We have several options for the implementation of our techniques. For rapid prototyping, initial implementation and experimentation, our recent Hipster system is a suitable starting point. We will also apply our results to reasoning about functional programs. As second step, we will therefore integrate the proposed techniques in the HipSpec system for reasoning about programs written in the Haskell language.

WP3A: Hipster. The advantage of working with Hipster and Isabelle is that there is already a very large proof library available for data mining. We can also use existing machine learning tools from the Sledgehammer system to quickly create first prototypes of the algorithms described above for initial experimentation and evaluation. While we do not work directly with source code in Isabelle, these kind of tools are still very useful to the formal methods community, as systems can *modelled* in Isabelle before being implemented. Verifying the model gives a higher level of assurance that the design is correct, and faulty design choices can be caught early, before implementation starts.

WP3B: HipSpec/Haskell. The HipSpec project works toward creating a verification tool for the Haskell language which is easy to use for regular programmers. We will therefore integrate the machine learning based techniques proposed here with HipSpec and the Haskell GHC compiler. This requires extensions to annotate Haskell code with properties proved by HipSpec and build up libraries of verified Haskell code.

WP4: Evaluation and Use Cases. Evaluation will be interleaved with development throughout the project. Suitable evaluation cases for the different WPs will be identified at the start of the project and larger case-studies performed towards the end of the project. Two such larger case studies are described below:

Java Virtual Machine Model. To verify complex systems, these are often modelled in some language more suitable for reasoning, for instance in proof assistants such as ACL2 or a functional programming language like Haskell. One such very ambitious project concerned modelling the Java Virtual Machine in the interactive proof assistant ACL2 [25]. This development contains many inductive proofs, which in ACL2 were performed interactively: the user had to invent and add intermediate lemmas until the main theorem could be proved by the system. We expect our system to find the appropriate lemmas and thus fully automate many of these proofs.

Haskell Libraries and Open Source Code. Verification can increase trust in program libraries. We will therefore apply our automated induction techniques to some of the standard libraries of Haskell. These libraries contain basic data-structures, such as lists, trees, maps etc, and are large enough to be challenging. Showing that we can verify commonly used libraries will have large impact on the Haskell community and will increase the trust in the code. Additional evaluation examples and case-studies can be sourced from *Hackage*, which is a large online repository of open source Haskell libraries for a wast range of applications.

Expected Final Results.

- Novel methods for lemma discovery using heuristic knowledge gathered from existing proof libraries, which will be better equipped to handle large theories than existing theory exploration algorithms. This will move the boundary for the size of theories which current systems can handle efficiently towards theories representing real-world programs.
- Experimental results on efficient feature extraction and learning algorithms for the theorem proving domain. These will improve and facilitate knowledge re-use of proof libraries.
- Tools for both Isabelle and the Haskell language that efficiently and automatically discover lemmas and prove user stated properties by induction.
- Verified Haskell libraries, illustrating the strengths and weaknesses of our approach and tools.

5. SIGNIFICANCE

Formal methods are increasingly important to avoid costly or dangerous situations as computers become even more widely used in all parts of society. The interest and significance of research into verified software is thus on the increase, as exemplified by the Verified Software Initiative, a 15-year international movement targeting large scale software verification [13]. Theorem provers are important tools for formal methods and program verification. However, automated support for proofs by induction is still largely missing. One of the main challenges in automating inductive proofs is discovery of auxiliary lemmas, which themselves may need inductive proofs. Our group has already developed the state-of-the-art inductive prover HipSpec, which took a new approach by integrating theory exploration for lemma discovery. In this project we go further by utilising information extracted by machine learning from existing proof libraries of previously formalised mathematics. This novel work will make current techniques more robust and scalable. This is crucial for applicability of the results in the context of program verification, where proofs and theories can be large and complex.

A specific application of the proposed research is verification of functional programs. The functional programming paradigm is becoming increasingly popular, with uptake by companies such as Ericsson (the Erlang and Feldspar languages) and Microsoft (the F# language). Functional programs are structured by recursion, and to prove properties about them, we need inductive proofs. Inductive proofs typically require considerable expertise, in this proposal we address one of the main challenges: automated lemma discovery. Advancing the state-of-the-art in inductive theorem proving will have broad impact on the verification community if we can equip programmers with easy to use automated tools for performing inductive proofs. There is industry interest; Microsoft Research funded a 3-month project for development of the first HipSpec prototype in 2012.

Our results will also have an impact on interactive systems, where programs and system designs might be modelled. Interactive systems typically also provide automatic techniques. We will both increase automatic support of inductive proofs, e.g. to provide hints of suitable lemmas, and improve support for knowledge re-use in mathematical proof libraries.

6. COLLABORATIONS

Local Collaborations at Chalmers. I am member of the Division of Software Technology at Chalmers University. The division house world-class research in functional programming, automated testing and verification. Within the Chalmers Computer Science division, there is further expertise in algorithms and machine learning. I bring expertise in inductive theorem proving and theory exploration, which was the topic of my PhD at the University of Edinburgh. This combination of skills makes an ideal environment for a project like this to succeed.

In particular, I will continue to work closely with the HipSpec group: Prof. Koen Claessen, Dr. Nicholas Smallbone and Dan Rosén, for whom I am also PhD co-supervisor. Dr. Laura Kovacs will be another local collaborator on the program verification side. She works with the state-of-the-art automated first-order theorem prover Vampire and is interested in extending this system with techniques for automated induction, with applications to reasoning about imperative programs with loops. On the machine learning side, I will work with Prof. Devdatt Dubhashi from the Machine Learning, Algorithms and Computational Biology Group.

External Collaborations. I visit the Mathematical Reasoning Group (MRG) at the University of Edinburgh and Heriot Watt University (also located in Edinburgh) for approximately six weeks per year as part of my VINNMER Marie Curie Fellowship project. The MRG conducts world leading research in the fields of proof-planning, automated inductive theorem proving and mathematical theory exploration. The leader of the MRG is Prof. Alan Bundy, who was the supervisor of my PhD. Other collaborators in Edinburgh include Dr. Jacques Fleuriot, who recently was awarded a grant on applications of machine learning to collaborative theorem proving. I also work with members of the AI for Formal Methods project, which in addition to members in Edinburgh include Prof. Cliff Jones and Dr Leo Freitas at the University of Newcastle. I have an ongoing collaboration with Dr. Ekaterina Komendantskaya and Dr. Jonathan Heras at the University of Dundee who are very interested in machine learning for interactive theorem proving. Our group also have good connections with Dr. Rustan Leino at Microsoft Researach, where our PhD student Dan Rosén will spend the summer for an internship.

REFERENCES

[1] A. Bundy. The use of explicit plans to guide inductive proofs. In *CADE*, pages 111–120, 1988.

[2] A. Bundy, D. Basin, D. Hutter, and A. Ireland. *Rippling: meta-level guidance for mathematical reasoning*. Cambridge University Press, New York, NY, USA, 2005.

[3] K. Claessen and J. Hughes. QuickCheck: a lightweight tool for random testing of Haskell programs. In *Proceedings of the fifth ACM SIGPLAN international conference on Functional programming*, ICFP '00, pages 268–279, New York, NY, USA, 2000. ACM.

[4] K. Claessen, M. Johansson, D. Rosén, and N. Smallbone. HipSpec: Automating inductive proofs of program properties. *IJCAR Workshop on Automated Theory eXploration (ATX 2012)*, 2012.

[5] K. Claessen, M. Johansson, D. Rosén, and N. Smallbone. Automating inductive proofs using theory exploration. In *Proceedings of the Conference on Auomtated Deduction (CADE)*, 2013.

[6] K. Claessen, N. Smallbone, and J. Hughes. QuickSpec: guessing formal specifications using testing. In *Proceedings of the 4th international conference on Tests and proofs*, TAP'10, pages 6–21, Berlin, Heidelberg, 2010. Springer-Verlag.

[7] S. Colton. *Automated Theory Formation in Pure Mathematics*. Springer Verlag, 2002.

[8] L. De Moura and N. Bjørner. Z3: an efficient SMT solver. In *Proceedings of TACAS*, TACAS'08/ETAPS'08, pages 337–340. Springer-Verlag, 2008.

[9] J. Denzinger and S. Schulz. Automatic Acquisition of Search Control Knowledge from Multiple Proof Attempts. *Inform. and Comput.*, 162(1-2):59–79, 2000.

[10] L. Dixon and J. D. Fleuriot. Higher order rippling in IsaPlanner. In *Theorem Proving in Higher Order Logics*, volume 3223 of *LNCS*, pages 83–98, 2004.

[11] L. Freitas and I. Whiteside. Proof patterns for formal methods. In *Proceedings of the Conference on Formal Methods (FM)*, 2014. To appear.

[12] J. Heras, E. Komendantskaya, M. Johansson, and E. Maclean. Proof-pattern recognition and lemma discovery in ACL2. In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 8312 of *LNCS*, pages 389–406. Springer, 2013.
[13] C. Hoare, J. Misra, G. T. Leavens, and N. Shankar. The verified software initiative: A manifesto. *ACM Comput. Surv.*, 41(4):22:1–22:8, Oct. 2009.
[14] A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16:16–1, 1995.
[15] M. Johansson, L. Dixon, and A. Bundy. Lemma discovery and middle-out reasoning for automated inductive proof. In S. Siegler and N. Wasser, editors, *Induction, Verification and Termination Analysis: Festschrift for Christoph Walther*, volume 6463 of *LNAI*, pages 102–116. Springer, 2010.
[16] M. Johansson, L. Dixon, and A. Bundy. Conjecture synthesis for inductive theories. *Journal of Automated Reasoning*, 47(3):251–289, 2011.
[17] M. Johansson, D. Rosén, N. Smallbone, and K. Claessen. Hipster: Integrating theory exploration in a proof assistant. Accepted for the Conference on Intelligent Computer Mathematics (CICM), 2014. To appear.
[18] M. Kaufmann, M. Panagiotis, and J. Strother Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, 2000.
[19] D. Kühlwein, J. C. Blanchette, C. Kaliszyk, and J. Urban. MaSh: Machine Learning for Sledgehammer. In *Proceedings of ITP'13*, LNCS, 2013.
[20] D. Kühlwein, T. van Laarhoven, and T. H. E. Tsivtsivadze, J. Urban. Overview and evaluation of premise selection techniques for large theory mathematics. In *IJCAR'12*, volume 7364 of *LNCS*, pages 378–392, 2012.
[21] K. R. Leino. Automating induction with an SMT solver. In *Proceedings of VMCAI*. Springer, 2012.
[22] M. T. Llano, A. Ireland, and A. Pease. Discovery of invariants through automated theory formation. *Formal Aspects of Computing*, 2012.
[23] R. L. McCasland, A. Bundy, and P. F. Smith. Ascertaining mathematical theorems. *Electron. Notes Theor. Comput. Sci.*, 151(1):21–38, Mar. 2006.
[24] O. Montano-Rivas, R. McCasland, L. Dixon, and A. Bundy. Scheme-based theorem discovery and concept invention. *Expert Systems with Applications*, 39(2):1637–1646, Feb. 2012.
[25] J. S. Moore. Proving theorems about Java and the JVM with ACL2. In *Models, Algebras and Logic of Engineering Software*, pages 227–290. IOS Press, 2003.
[26] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
[27] L. Paulson and J. Blanchette. Three years of experience with Sledgehammer, a practical link between automation and interactive theorem provers. In *Proceedings of IWIL-2010*, 2010.
[28] D. Rosén. Proving equational Haskell properties using automated theorem provers. Master's thesis, University of Gothenburgh, 2012.
[29] W. Sonnex, S. Drossopoulou, and S. Eisenbach. Zeno: An automated prover for properties of recursive datatypes. In *Proceedings of TACAS*, pages 407–421. Springer, 2012.
[30] G. Tassey. The economic impacts of inadequate infrastructure for software testing. Technical report, National Institute of Standards and Technology, 2002.
[31] J. Urban, G. Sutcliffe, P. Pudlak, and J. Vyskocil. MaLARea SG1 - Machine learner for automated reasoning with semantic guidance. In *IJCAR'08*, volume 5195 of *LNCS*, pages 441–456, 2008.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2014-42285-115143-47

Name of applicant
Johansson, Moa

Date of birth
810720-8962

Title of research programme
Learning and Exploration in Automated Theorem Proving

Appendix B

Curriculum vitae

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

B CV: Moa Johansson

B.1 Undergraduate Degree

Bachelor of Science with Honours in Artificial Intelligence and Computer Science, University of Edinburgh, UK, 2001-2005.

B.2 Postgraduate Degree

PhD in Informatics, University of Edinburgh, UK, 2005-2009.

- Title: *Automated Discovery of Inductive Lemmas*.
- Supervisors: Prof. Alan Bundy and Dr. Lucas Dixon.

B.3 Post-Doctoral Position

University of Verona, Italy, 2009-2011.

The project concerned interpolation techniques for first-order theorem provers using the superposition calculus.

B.4 Current Position

Research Fellow at Chalmers University since October 2011.

I hold a VINNMER Marie Curie fellowship entitled *Reasoning about Recursive Programs* until March 2015. The position includes 80% research and 20% teaching.

B.5 Previous Employment

Research Associate, University of Edinburgh, July-August 2005.

B.6 Research Supervision of Doctoral Students

Assistant supervisor for PhD student Dan Rosén, who started in January 2013.

B.7 Absences

N/A

B.8 Additional Information

Other Research Grants

I currently hold a VINNMER Marie Curie Fellowship (Oct 2011 - March 2015). The fellowship funds 50% of my salary.

Professional Activities

I am very active in the automated induction and theory exploration communities. I have organised several workshops on these topics, both formal and informal. Most recently I was part of the organisation of a well attended international workshop about proofs by induction at Imperial College in London, which we hope will turn into a regular event as interest in the topic seems to be on the increase.

An overview of other activities are listed below:

- Co-chair and PC Member of the Workshop on Automated Mathematical Theory Exploration (AUTOMATHEO) 2014, to be held in conjunction with the Federated Logic Conference in Vienna this summer.
- Organiser of an workshop on Automated Induction at Imperial College in London in November 2013.
- Organiser of the CIAO workshop on automated reasoning at Chalmers in 2012.
- Program Committee Member of the Workshop on Invariant Generation (WING) 2012.
- Program Committee Member of the Mexican International Conference on Artificial Intelligence (MICAI) 2010, 2011.
- Program Committee Member of the Workshop on Automated Mathematical Theory Exploration (AUTOMATHEO) 2010.
- Co-chair of the Isabelle Workshop 2007.
- External reviewer for a variety of conferences, including
  - Conference on Automated Reasoning (CADE)
  - Conference on Computer Aided Verification (CAV)
  - International Joint Conference on Automated Reasoning (IJCAR)
  - Conference on Software Engineering using Formal Methods (SEFM)

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2014-42285-115143-47

Name of applicant
Johansson, Moa

Date of birth
810720-8962

Title of research programme
Learning and Exploration in Automated Theorem Proving

Appendix C

List of publications

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

C Publications: Moa Johansson

Note to non computer scientists: Please bear in mind that conference articles in computer science are the most common form of refereed publication, as they are peer reviewed full articles (not 1-2 page abstracts) with high impact factor.

C.1 Journal Articles

- * *Conjecture Synthesis for Inductive Theories*. Moa Johansson, Lucas Dixon and Alan Bundy. Journal of Automated Reasoning 47(3), p. 251-289, 2011.

C.2 Articles in refereed collections and conference proceedings

- * *Hipster: Integrating Theory Exploration in a Proof Assistant*. Moa Johansson, Dan Rosén, Nicholas Smallbone and Koen Claessen. Accepted for the Conference on Intelligent Computer Mathematics. To appear in 2014.
- * *Proof Pattern Recognition and Lemma Discovery in ACL2*. Jonathan Heras, Ekaterina Komendantskaya, Moa Johansson, and Ewen Maclean. Proceedings of the Conference on Logic for Programming, Artificial Intelligence, and Reasoning, 2013. Springer LNCS 8312, p.389-406.
- * *Automating Inductive Proofs using Theory Exploration*. Koen Claessen, Moa Johansson, Dan Rosén and Nicholas Smallbone. Proceedings of the Conference on Automated Deduction, 2013. Springer LNCS 8312, p. 392-406.
- *A Calculus for Conjecture Synthesis*. Moa Johansson, Lucas Dixon and Alan Bundy. Post-proceedings of the Workshop on Automated Mathematical Theory Exploration (AUTOMATHEO). IFCoLog Conference Proceedings Series. Accepted for publication, in press.
- *The Theory Behind TheoryMine*. Alan Bundy, Flaminia Cavallo, Lucas Dixon, Moa Johansson and Roy McCasland. Post-proceedings of the Workshop on Automated Mathematical Theory Exploration (AUTOMATHEO). IFCoLog Conference Proceedings Series. Accepted for publication, in press.
- *HipSpec: Automating Inductive Proofs of Program Properties*. Koen Claessen, Moa Johansson, Dan Rosén and Nicholas Smallbone. Workshop on Automated Theory Exploration (ATX), at the International Joint Conference of Automated Reasoning, 2012.
- *On Interpolation in Decision Procedures*. Maria Paola Bonacina and Moa Johansson. Proceedings of the International Conference on Automated Reasoning with Anlytic Tableaux and Related Methods, 2011. Springer LNAI 6793, p.1-6.

- *Towards Interpolation in an SMT-solver with Integrated Superposition*. Maria Paola Bonacina and Moa Johansson. Satisfiability Modulo Theory (SMT) Workshop 2011.
- ** Case-Analysis for Rippling and Inductive Proof*. Moa Johansson, Lucas Dixon and Alan Bundy. Proceedings of the Interactive Theorem Proving Conference (ITP), 2010. Springer LNCS 6172, p. 291-306.
- *Dynamic Rippling, Middle-Out Reasoning and Lemma Discovery*. Moa Johansson, Lucas Dixon and Alan Bundy. Verification, Induction and Termination Analysis, 2010. Springer LNCS 6463, p.102-116.
- *Best-First Rippling*. Moa Johansson, Lucas Dixon and Alan Bundy. Reasoning Action and Interaction in AI Theories and Systems - Essays Dedicated to Luigia Carlucci Aiello, p. 83-100, Springer, 2006. Also presented at the STRATEGIES workshop at FLoC 2006 in Seattle.

C.3 Freely Available Computer Programs

- **IsaPlanner** is a proof planning system developed at the University of Edinburgh. The main use of IsaPlanner is as an automated theorem prover for inductive proofs. I am one of the developers.
- **IsaCoSy** is a theory exploration system which automatically attempts to find mathematical theorems in a given theory. It uses IsaPlanner to prove conjectures it finds. I developed IsaCoSy as part of my PhD.
- **HipSpec** is an inductive theorem prover and theory exploration system. I am part of the group developing HipSpec, its main developer is my PhD student, Dan Rosén.
- **Hipster** is a new theory exploration tool allowing users of the interactive proof assistant Isabelle to get lemma suggestions for stuck proofs. It uses HipSpec as a backend.

IsaPlanner and IsaCoSy are available for downloading from
`http://dream.inf.ed.ac.uk/projects/isaplanner`.

HipSpec is available from `https://github.com/danr/hipspec`.

Hipster is available from `https://github.com/moajohansson/IsaHipster`.

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2014-42285-115143-47

Name of applicant
Johansson, Moa

Date of birth
810720-8962

Title of research programme

Appendix N

Budget and research resources

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

N Budget

I apply for 80% of my salary at the level of *Forskarassistent* for four years (the remaining 20% are covered by other funding—mainly teaching).

Cost (tkr)	2015	2016	2017	2018	2019
Moa Johansson, salary 80%	471	648	670	692	179
Travel and conference costs	30	30	30	30	
Direct operating costs, computers, IT etc.	99	90	92	94	17
Total:	736	967	999	1032	259

VETENSKAPSRÅDET
THE SWEDISH RESEARCH COUNCIL

Kod
2014-42285-115143-47

Dnr
2014 -

Name of applicant
Johansson, Moa

Date of birth
810720-8962

Reg date
2014-04-09 15:30:43

Project title
Learning and Exploration in Automated Theorem Proving

NT

Research grants NT April 9, 2014
Project Grant Junior Researchers
*Övrigt generellt
Chalmers tekniska högskola

NT-2

Appendix S

Signatures

2015	2016	2017	2018	2019	
736	967	1000	1031	258	2015-04-01 -- 2019-03-31

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
- International Recruitment

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

Vetenskapsrådets noteringar
Kod

2014-42285-115143-47

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