Appendix A – Research Programme

GenPro: Generating and Proving Program Properties via Symbol Elimination

1. Purpose and Aims

Software systems used in our daily life, such as networking, security, autonomous devices, traffic control, etc., heavily rely on software used in them. Such software is becoming increasingly more sophisticated, resulting in system malfunctioning and error-prone and insecure system behavior. Software errors are very costly. There are many studies attempting to quantify the cost of software failures, including survey reports by the European Services Strategy Unit, the British Computer Society, and the US National Institute of Standards and Technology. For example, according to US National Institute of Standards and Technology software errors cost the US economy nearly 60 billion annually, and 80% of development costs involve identifying and correcting errors. For this reason, there is a growing interest, both industrial and academic, in applying formal methods for ensuring reliability of long-lived, high-quality software systems.

Formal verification aims at providing a methodology that produces more reliable and robust systems. Companies developing and running safety-critical systems, such as Microsoft, IBM, and Intel, have therefore started to use precise formal methods and scalable tools developed in this area. Precision of designed methods is necessary in order to ensure that the reported potential system errors are indeed errors, and that no bugs are omitted during the verification process. Scalability is required so that the tools work efficiently for large systems. Among the main techniques of formal verification are model checking, abstract interpretation, satisfiability modulo theory (SMT) reasoning, and first-order theorem proving. These methods are interrelated and many modern formal verification tools use a combination of them.

The objective of our proposal is to develop new methods advancing the applicability of first-order theorem proving in formal software verification. The project will explore and advance the power of the symbol elimination method, introduced recently by ourselves, for generating and proving program properties. Symbol elimination is based on first-order theorem proving and is fully automatic. It was the first ever method able to automatically discover complex program properties with quantifier alternations. The method is based on the following steps. Given a program loop, we first extend the language of the program with additional symbols, such as a loop counter. Next, we extract various information about the program that can be expressed by first-order formulas. These formulas can however use the introduced auxiliary symbols. Therefore, we run a superposition-based first-order theorem prover to eliminate the auxiliary symbols and obtain program properties expressed in the original language of the program.

In our project we will pay special attention to developing new theory and tools based on symbol elimination, solve automated reasoning problems arising in program verification, and thus prove automatically the validity of safety properties of software.

Safety properties ensure that something ”bad” never happens in the program. Verifying such properties becomes an especially challenging task when programs contain (nested) loops or recursion. The verification of such programs needs additional information, so-called program assertions, that express conditions to hold at certain intermediate points of the program. Typical auxiliary assertions are loop invariants, which describe program states that can be reached during program computations and thus are essential for safety property verification. The effectiveness of using symbol elimination in formal verification thus crucially depends on whether such assertions, even trivial ones, can be deduced automatically.

The **overall purpose** *of our project is to design new, unconventional methods for reasoning about program properties, by using symbol elimination in first-order theorem proving. We will implement world-leading tools that are likely to be used by others in the area, and apply our methods and tools on problems of industrial relevance.* The results of our project will provide fundamentally new ways of generating and proving program properties by symbol elimination,

and have a substantial impact both on the theory and practice of program verification. To achieve the aim of the project, the **specific goals of our project** includes the following three directions:

1. Developing new methods for *generating complex loop invariants* using symbol elimination. This research will include extensions of symbol elimination to reason about loops with unbounded data types such as arrays, lists, pointers, and heaps, and generate quantified invariants, possibly with quantifier alternations, fully automatically. We believe that the results will be highly original and ground-breaking, since the research we propose is the only method for property generation based on consequence finding by symbol elimination in first-order theorem provers.
2. Designing new techniques for *constructing small Craig interpolants* from proofs in full first-order logic with theories, and derive interpolants over complex data types. Craig interpolants will be further used in the process of invariant generation and model checking. We will develop efficient interpolation algorithms using symbol elimination in first-order theorem proving and apply our interpolation algorithms both for first-order and SMT reasoning. Our results will thus not be limited to reasoning in ground (that is, quantifier-free) theories, but provide automated methods to derive interpolants in quantified first-order theories, such as the theory of arrays, lists, pointers, and heaps.
3. Developing and implementing *efficient reasoning methods with theories and quantifiers*. Further progress in program verification needs efficient methods and tools for analyzing complex data structures, pointers and memory allocation. Reasoning with quantifiers and theories is the main ingredient required for a major breakthrough making such analysis feasible. We therefore expect that this research direction will significantly advance the use of theorem provers in verification. In particular, we will focus on theory reasoning in first-order superposition calculus and instantiation-based theorem proving, and design new ways of cooperation between first-order provers and SMT solvers.

The proposed research directions will be accompanied by the development of the award-winning first-order theorem prover Vampire, to which the PI of the project is actively contributing. In particular, we will implement new extensions of Vampire to support generating and proving program properties. The new features of Vampire will be *applied and evaluated on academic and industrial programs*, for the latter we have informal agreements about collaboration with verification teams at Microsoft Research and Intel.

We expect our project to have a deep and long-lasting impact in reasoning-based formal verification, yielding novel methods and scientific breakthroughs in using symbol elimination for software verification. The timeliness of the proposed research is witnessed by the growing academic and industrial interest in formal verification, which is due to availability of very powerful tools and methods based on previous research in the area.

2. Survey of the Field

We overview the state-of-the-art in invariant generation, interpolation, and reasoning in first-order theories. We give key references to existing methods and discuss the novel features of our project in relation to these methods. Due to the page limit, we do not discuss the wide range of tools in this area.

2.1 Quantified Invariant Generation. Invariants with quantifiers are important for the verification of programs over non-numeric data structures, such as arrays, due to the unbounded nature of array structures. Such invariants can express relationships among array elements and properties involving arrays and scalar variables of the loop, and thus significantly ease the verification task. Automated discovery of array invariants is therefore a challenging topic.

In [36, 10] loop invariants are inferred by predicate abstraction over a set of a priori defined predicates, while [13] employs constraint solving over invariant templates. A different approach

is given in [33], where input predicates in conjunction with interpolation are deployed to compute invariants. Unlike these techniques, in [30] we described a framework for automatically inferring array invariants without any user guidance, by introducing the symbol elimination method in first-order theorem proving. User guidance is also not required in [14, 4], but invariants are derived by abstract interpretation over array indexes [14] and array segments [4]. However, these approaches can only infer universally quantified invariants. Our approach in [30] is fundamentally different from the afore-cited techniques. The use of symbol elimination in quantified first-order theories has not yet been addressed by other methods. Based on our previous work [30, 20], we will provide an automatic method for generating quantified invariants, including those with quantifier alternations, which cannot be handled by other techniques.

2.2 Interpolation in Formal Verification. Interpolation [5] is an important technique in verification and static analysis of programs, in particular in invariant generation and bounded model checking, see e.g. [23, 33]. The key ingredient of theorem proving based interpolation is the notion of a so-called local proof, since local proofs admit efficient interpolation algorithms – see [32] and our work [31].

There are several interpolant generation algorithms for various theories. For example, [23, 3] derive interpolants from resolution proofs in the ground theory of linear arithmetic and uninterpreted functions. The approach described in [35] generates ground interpolants in the combined theory of arithmetic and uninterpreted functions using constraint solving techniques over an a priori defined interpolant template. The method presented in [33] computes quantified interpolants from first-order resolution proofs over scalars, arrays and uninterpreted functions. Craig interpolation has recently been generalized to so-called tree interpolants for their use in concurrent [11] and recursive [15] programs. Results of [12] can however be applied only for interpolation in the quantifier-free theory of uninterpreted functions and linear arithmetic. In contrast to these techniques, in [31] we gave an algorithm for interpolant extraction from local proofs in any sound calculus, including calculi for various theories, such as arithmetic or a theory of arrays. Thus, our interpolation method in [31] is not limited to decidable theories for which interpolation algorithms are known. However, a consequence of this generality is that, unlike [23, 3, 35, 11, 15], in [31] we do not guarantee finding interpolants even for decidable theories or for theories where interpolants are known to exist. In [22] we addressed qualitative aspects of interpolation, and gave a general framework to minimize interpolants wrt their size and number of quantifiers. Our work in [22] was the first method providing a flexibility in computing interpolants of different strength and structure.

2.3 Reasoning with Theories and Quantifiers. For reasoning about properties involving both quantifiers and theories, SMT solvers and first-order theorem provers are complementary. SMT solvers, see e.g. Z3 [7] and Yices [8], are powerful when it comes to proving formulas without quantifiers in combinations of common first-order theories such as linear arithmetic, arrays and uninterpreted functions, but are inefficient for reasoning in full first-order logic. On the contrary, first-order theorem provers, see e.g. Vampire [19, 21] and iProver [26] are very efficient in working with quantifiers, but have essentially no support for theory reasoning.

Combining first-order proving and theory reasoning is very hard. For example, some simple fragments of combining first-order reasoning with linear arithmetic are already Π^1_1-complete [27]. Most of the modern systems are based on two approaches to such reasoning: trigger-based (heuristic) ground instantiation of quantified axioms in SMT solvers [7, 8] and (incomplete) axiomatization of theories in first-order provers [30, 20]. A tighter integration is described in [6, 1]. Motivated by our previous results using Vampire [30, 20], we will investigate how to integrate theory reasoning with superposition-based first-order theorem proving and implement our results in Vampire. We believe that our project will enable solving problems beyond the reach of other methods.

3. Project Description

3.1 Methodology. To formally verify computer programs means to apply mathematical arguments to prove the correctness of systems. Since systems have bugs, formal verifications aims at finding and correcting such bugs. There are theoretical results showing undecidability of almost every problem related to program correctness, in particular to reason and prove properties about the logically complex part of the code characterized by (nested) loops and recursion.

The practice of designing new tools supporting the verification of program with loops meets very hard everyday challenges, and faces the problem of generating and reasoning about loop properties, in particular loop invariants. Providing loop properties manually requires a considerable amount of work by highly qualified personnel and thus often makes verification prohibitively expensive. Therefore, generation of program properties without user-provided annotations is invaluable in making program analysis and verification economically feasible.

In our project we will focus on developing methods for the automated synthesis of loop properties of complex programs with unbounded data structures, such as arrays, lists, pointers, and heaps. First, we will focus on the generation of loop invariants. Next, as Craig interpolation provides an alternative way to reason about loop invariants, we will compute interpolants and use them further in invariant synthesis. Finally, as loop properties over unbounded data types involve quantifiers and theory symbols, we will address reasoning in full first-order (FO) logic with theories. The common method of our project is the *symbol elimination method*, introduced by ourselves in [30].

In a nutshell, symbol elimination is based on the following ideas. Suppose we have a program P with a set of variables V. The set V defines the language of P. We extend the language of P to a richer language V^+ by adding functions and predicates, such as loop counters. After that, we automatically generate a set Π of FO properties of the program in the extended language V^+, by using techniques from symbolic computation and theorem proving. These properties are valid properties of the program, however they use the extended language V^+. Then we derive from Π program properties in the original language P, thus "eliminating" the symbols in $V^+ \setminus V$.

The distinctive feature of the symbol elimination method is its power to automatically generate, with no user guidance, statements expressing complex computer program properties, including those with quantifier alternations. The method uses, in a new way, the power of a superposition-based FO theorem prover and symbolic computation to derive program properties that hold at intermediate points of the program. Such properties are crucial to ensure the safety of computer systems.

To illustrate the need of quantified loop properties, consider the program given in Figure 1, written in a C-like syntax. The program fills an array B with the non-negative values of a source array A, and an array C with the negative values of A. It is not hard to derive that at after any iteration n of this loop (assuming $0 \leq n \leq k$), the linear invariant relation $a = b + c$ holds. For example, this property can be derived by the methods of [34, 29]. However, such a property does not give us much information about the arrays A, B, C and their relationships. For example, one may want to derive the following properties of the loop (n denotes the loop counter):

```
a := 0; b := 0; c := 0;
while (a ≤ k) do
  if A[a] ≥ 0
    then B[b] := A[a];b := b + 1;
    else C[c] := A[a];c := c + 1;
  a := a + 1;
end do
```

Figure 1: Array partition.

1. Each of $B[0], \ldots, B[b-1]$ is non-negative and equal to one of $A[0], \ldots, A[a-1]$. Formulating this property in FO logic, yields the loop invariant:

$$(\forall p)(0 \leq p < n \implies B[p] \geq 0 \ \wedge \ (\exists k)(0 \leq k < a \wedge A[k] = B[p])).$$

2. For every $p \geq b$, the value of $B[p]$ is equal to its initial value. Writing it in FO logic, this loop invariant is:
$$(\forall p)(p \geq b \implies B[p] = B_0[p]),$$
where B_0 denotes the value of B before the first iteration of the loop.

These loop properties in fact describe much of the intended function of the loop and can be used to verify properties of programs manipulating arrays in which this loop is embedded. Note however that the first invariant, when formulated in first-order logic, requires quantifier alternations, whereas the second property involves only universal quantification. Generating such properties requires reasoning in full FO logic with theories, in our example in the FO theory of arrays and integers.

This proposal addresses the quest of generating such quantified program properties. We aim at improving our symbol elimination method for generating quantified loop invariants and interpolants for programs with complex flow and unbounded data structures, such as arrays, lists, pointers, and heaps. As reasoning about such programs requires reasoning with both theories and quantifiers, our project will also address proving in FO logic with quantified theories. Our project will thus be structured in three working packages (WP):

(WP 1). Inferring quantified invariants in theories with unbounded data types;
(WP 2). Generating quantified interpolants from proofs in such theories;
(WP 3). Efficient reasoning with theories and quantifiers.

These work packages are strongly related and depend on each others' results. For example, properties generated in (WP1) and (WP2) will be used for testing (WP3), and any development in (WP3) will improve the results of (WP1)-(WP2). In what follows, we describe the proposed research for each work package, along with their deliverables and milestones.

WP1. Quantified Loop Invariant Generation. In (WP1) we plan to address the following five aspects of symbol elimination:
(WP1.1) We will first design various *FO theories of data types*. For doing so, auxiliary predicates expressing properties over the content of arrays, lists, pointers, and heaps will be added. Reasoning in FO theories of such unbounded data types will be necessary for extending symbol elimination to derive quantified invariants. (WP1.1) will use results of (WP3) and help (WP3) by providing hard benchmarks for reasoning with quantifiers and theories under study.
(WP1.2) We will extend symbol elimination with *generation of various classes of clause sets with eliminated symbols*. One interesting extension comes with inferring a minimized set of invariants to avoid redundant clauses that imply each other.
(WP1.3) Further development of symbol elimination will also require *analyzing programs with nested loops and complex arithmetic* followed by or interleaved with theorem proving and symbolic computation. For such analysis, we will apply advanced recurrence solving techniques from [29] together with the FO interpolation results of (WP2), as well as with the best existing static analysis methods and tools, for example [36, 14, 4].
(WP1.4) An open challenge of (WP1) is to show when symbol elimination is (or can be made) a *complete approach for invariant generation*. By completeness we mean that if a program has a quantified invariant of a certain form, our approach will find invariants implying them. We will also generalize (WP2) to *generate interpolants from which inductive invariants are inferred*.
(WP1.5) We will *extend Vampire* to analyze complex programs over various data types, and generate invariants using symbol elimination. Our methods will be *tested on benchmarks*, including large industrial examples coming from Intel and Microsoft (see our collaboration list).
Milestones and Deliverables:

- first-order theories of data types;

- new and complete invariant generation methods in such theories using symbol elimination;
- a first-order theorem prover with invariant generation;
- case studies of invariant generation for large programs.

WP2. Generating Quantified Interpolants. (WP2) will be structured on the following five applications of symbol elimination:
(WP2.1) We will use symbol elimination in FO theorem proving and design new methods to *extract interpolants from arbitrary FO superposition proofs*. In particular, we will address interpolation in FO non-local proofs.
(WP2.2) Using the theorem proving engine of (WP3), we will design efficient superposition algorithms for *generating interpolants in the combination of various FO theories*, such as arithmetic, unbounded data structures and uninterpreted function symbols. Various methods of *minimizing interpolants* will also be studied, possibly based on proof transformations.
(WP2.3) We will use generalized Craig interpolants, called tree interpolants [11, 15], and *compute FO tree interpolants for concurrent programs*.
(WP2.4) We will *extend Vampire* with efficient interpolation algorithms for complex programs, and make Vampire work together with model checkers, such as CPAChecker [2]. Our results will be evaluated on various benchmarks, including examples coming Intel and Microsoft (see our collaboration list).
(WP2.5) A the quality of interpolation algorithms can only be evaluated in the context of practical software verification, our results will be *integrated in concrete verification tools*. We will experiment how the quality of minimized interpolants effects the efficiency of interpolation-based verification methods, in particular invariant generation, as discussed in (WP1).
Milestones and Deliverables:

- new methods of interpolation in FO theories using symbol elimination;
- an interpolating theorem prover for first-order formulas with data types;
- integration into model checking based verification tools;
- case studies of interpolation on concrete programs of industrial relevance.

WP3. Efficient Reasoning with Theories and Quantifiers. (WP3) will consist of the following four research directions:
(WP3.1) We will first add *existing SMT algorithms* for the theory of integers, reals and arrays to Vampire, study theory behavior and understand the best ways of using them within a FO theorem prover. After that we are interested in *new, less understood, theories*, such as those of pointers, queues and heaps.
(WP3.2) We will next extend FO theorem provers with *sound but incomplete theory axiomatizations*. In particular, using testcases from (WP1) and (WP2), we will study necessary and sufficient sets of theory axioms from which interesting program properties can be derived in extensions of the superposition calculus. We will need to understand what are the best simplification rules, literal selections, variable orderings and maybe also add some theory rules.
(WP3.3) We will address the *instantiation based theorem proving* framework of [9, 26], and use (WP3.1) for proving ground instanced of a FO problem. Proofs produced by (WP3.1) will be analyzed with the purpose of generating proof certificates. The proof certificates will be propagated back to the FO theorem prover, and new ground instances will be generated. However, understanding what parts of the proof are relevant is a non-trivial task, and requires a deeper understanding of how the SMT engine and the FO prover can work together.
(WP3.4) Our work will be based on *empirically driven development* of methods, resulting in improvements in algorithms and their use and combination in Vampire. These improvements will be evaluated on examples coming from (WP1) and (WP2), as well on industrial benchmarks coming from Microsoft and Intel (see our collaboration list).
Milestones and Deliverables:

- theory reasoning in superposition calculus;
- theory reasoning in instantiation-based theorem proving;
- new reasoning methods combining first-order provers and SMT solvers;
- case studies on academic and industrial examples.

3.2. Timetable and Organization. The project will last 5 years and will be led by Laura Kovács (PI). The work will be carried out by the PI and a PhD student under her supervision. There will be weekly project meetings to exchange ideas and discuss the progress.

The work plan of the project is illustrated in Table 2 and is is designed according to the dependencies among WPs. (WP1) and (WP2) crucially depend on (WP3), and therefore work in (WP3) is planned during the entire phase of the project. The work on (WP1) will start already in year 1, yielding results that can improve the research of (WP2) in year 2. Results of all WPs will be joined and tested in year 3 of the project, identifying this way the essential research directions for years 4 and 5. The working plan in years 4 and 5 is similar as in years 1 and 2.

Work Package	Year1	Year2	Year3	Year4	Year5
WP1	✓		✓	✓	
WP2		✓	✓		✓
WP3	✓	✓	✓	✓	✓

Figure 2: The GenPro Work Plan.

Our results will be disseminated as follows. We will publish articles in leading journals and conferences. We will develop the world-leading theorem prover Vampire. We will participate in tool competitions in the area. We will al organize workshops and tutorials at top international conferences, and scientific seminars at Chalmers.

4. Significance

Analyzing and verifying large computer programs requires non-trivial automation. Automatic generation and proving program properties is a key step to such an automation. Reasoning about program properties becomes an especially challenging task for programs with complex flow and, in particular, with loops. Further progress in reasoning about software therefore crucially depends on the existence of efficient methods and tools analyzing programs with nested loops and complex data structures. Our project addresses this challenge, and brings new non-standard approaches to generate and prove program properties via symbol elimination in first-order theorem proving. Our project will give unique, novel and significant contributions for the following reasons:

- Symbol elimination offers a fully automatic framework to generate invariants with arbitrary quantifier alternations, without using user/provided templates and annotations. By extending the application of symbol elimination to more complex programs and unbounded data types, our project will automatically infer properties that cannot be generated by other approaches.
- Generating quantified loop invariants using theorem provers is a new research area, initiated by our work in [30] over programs with integers and arrays. No technique is known to infer invariants over other data structures or theories by using theorem proving.
- Interpolation was mainly studied and implemented in the context of quantifier-free theories, see e.g. [23, 3, 11]. Our results will however target interpolation in fist-order theories, and will not limited to decidable theories. That is, results of our project can even be applied to theories for which no interpolation algorithms are known.
- We will design and implement efficient methods combining superposition theorem proving and SMT reasoning. We expect that our methods will be as efficient as first-order provers in reasoning with quantifiers and as efficient as SMT solvers in ground theory reasoning, solving thus some problems that are beyond the scope of existing technologies.

The proposed research for a PhD student is in the rapidly developing area of formal verification. This area attracts a lot of interest in industry, which means that successful PhD students can be employed by industry and/or apply their results in an industrial setting. The project will use and design rigorous techniques in computer science (formal methods), mathematics (combinatorics and computer algebra), and logic (reasoning and decision procedures), to support systems engineering methods and tools. Our project will thus also encourage interactions with mathematicians, logicians, and computer scientists.

5. Preliminary Results

Our work [30] on symbol elimination in theorem proving resulted in the first ever method to generate complex loop invariants with quantifier alternations. Symbol elimination, unlike other known methods, is completely automatic and requires no annotation or patterns. When applying symbol elimination on industrial safety-critical applications, in [20] we have shown that the loop properties obtained by symbol elimination could replace human-produced annotations in over 80% of test cases. The impact of symbol elimination method in formal verification is witnessed by the number of citations of our 2009 paper [30]: 63 international citations since 2009.

Since recent results on Craig interpolation provide an alternative approach to invariant generation, we extended symbol elimination to interpolation and gave a new algorithm for building interpolants from first-order local proofs [31]. Further, we designed a new method for computing small interpolants [22], by considering proof transformations and encoding the interpolant minimization problem as a pseudo-boolean optimization problem. The evaluation of our method on bounded model checking examples shows that minimization considerably decreases the interpolant size [18].

We also investigated the use of symbol elimination in symbolic computation and gave a complete algorithm for generating all polynomial invariants, by using symbol elimination in Gröbner basis computation [29]. In addition, when extending symbol elimination to quantifier elimination methods, our results in [16] gave an automated approach to derive linear inequality invariants. Interestingly, the application of symbol elimination in symbolic computation turned out to be a useful method for the timing analysis of real-time system [24].

On the implementation side, we made the Vampire theorem prover into an interpolating first-order prover with invariant generation [19, 21]. The PI of this project started her work on Vampire in August 2008. As a result of the PIs involvement in the development of Vampire, starting from 2009 the number of Vampire users has significantly increased. For example, in 2010–2011 Vampire was downloaded more than 1,000 times (counting only real downloads, with verified email addresses), that is, over 10 times a week.

We also designed the symbolic computation engine Aligator to derive polynomial invariants [28, 17]. Our work in using symbol elimination for the timing analysis of software has materialized in the r-TuBound software package [25].

Summarizing, I believe that my preliminary results of symbol elimination in formal verification provide a solid background for further progress in the area, addressing the challenge of reasoning about software with complex flow and various data types. My academic contribution and my experience in developing reasoning-based verification methods and tools enables me to make significant developments in both theory and implementation and ensures that the project successfully meets its scientific goals.

6. National and International Collaborations

We plan to collaborate with well-known researchers in the area, as listed below.

Chalmers. Starting with April 2013, I have joined Chalmers as an associate professor. My appointment to Chalmers was made with a purpose of building a competitive research group in formal methods and strengthen Chalmers' research expertise and international scientific competitiveness in formal verification. Topics of the current project are in the center of interest of several

researchers at Chalmers, including top scientists such as Prof. Wolfgang Ahrendt, Prof. Koen Claessen, Dr. Moa Johansson, Prof. Andrei Sabelfeld, and Prof. Geraldo Schneider. Our project will thus encourage cross-institutional collaboration at Chalmers and will benefit from the research expertise and the infrastructure of the department. To be more precise, our collaboration plans include the following actions: collaboration with Prof. Ahrendt and Prof. Schneider on invariant generation and verification tools (WP1)-(WP2), joint work with Prof. Claessen and Dr. Johansson on theorem proving (WP3), and collaboration with Prof. Sabelfeld on reasoning about properties over concurrent data structures such as queues and heaps (WP1).

Austrian RiSE. Together with 8 other researchers, I am a founding member of the Austrian Society for Rigorous Systems Engineering (ARiSE). The current project naturally targets collaboration with the other members of ARiSE, in particular with Prof. Thomas A. Henzinger (IST Austria) and Prof. Helmut Veith (TU Vienna) on (WP1) and (WP2); and with Prof. Armin Biere (JKU Linz) on (WP3).

The University of Manchester, UK. Our main results on symbol elimination in first-order theorem proving and Vampire were obtained in collaboration with Prof. Andrei Voronkov at the University of Manchester. We are keen to continue this collaboration on all parts of the project – (WP1), (WP2), and (WP3). Joint work is also scheduled during the entire development of Vampire. The collaboration between the PI and Prof. Voronkov has started in 2008, yielding several joint publications that are of direct relevance to the proposed project.

Industrial Collaborators. We propose joint work with Dr. Nikolaj Bjørner from Microsoft Research, US on (WP1) and (WP3), as well as with Dr. Zurab Khasidashvili from Intel Haifa on (WP1). Industrial examples coming from Microsoft and Intel will be used to evaluate the results of our project. The PI has recently visited Microsoft Research Redmond in February 2013 with the aim of starting deeper collaboration and joint work. The PI already consulted Intel in December 2010, and has recently restarted her consulting activities with Intel.

7. Other Grants

I have joined Chalmers as an associate professor in April 2013. The current application is my first VR proposal, and I do not apply for other VR grants. The VR grant would make it possible for me to start my own independent research group in formal methods at Chalmers. The VR grant would also provide me the opportunity to integrate myself in the Swedish scientific community.

Literature

[1] L. Bachmair, H. Ganzinger, and U. Waldmann. Refutational Theorem Proving for Hierarchic First-Order Theories. *Appl. Algebra Eng. Commun. Comput.*, 5:193–212, 1994.

[2] D. Beyer and M. E. Keremoglu. CPAchecker: A Tool for Configurable Software Verification. In *Proc. of CAV*, pages 184–190, 2011.

[3] A. Cimatti, A. Griggio, and R. Sebastiani. Efficient Interpolant Generation in Satisfiability Modulo Theories. In *Proc. of TACAS*, pages 397–412, 2008.

[4] P. Cousot, R. Cousot, and F. Logozzo. A Parametric Segmentation Functor for Fully Automatic and Scalable Array Content Analysis. In *Proc. of POPL*, pages 105–118, 2011.

[5] W. Craig. Three uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory. *J. Symbolic Logic*, 22(3):269–285, 1957.

[6] L. de Moura and N. Bjørner. Engineering DPLL(T) + Saturation. In *Proc. IJCAR*, pages 475–490, 2008.

[7] L. de Moura and N. Bjorner. Z3: An Efficient SMT Solver. In *Proc. of TACAS*, pages 337–340, 2008.

[8] B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *Proc. of CAV*, pages 81–94, 2006.

[9] H. Ganzinger and K. Korovin. Theory Instantiation. In *Prof. of LPAR*, pages 497–511, 2006.

[10] S. Gulwani, B. McCloskey, and A. Tiwari. Lifting Abstract Interpreters to Quantified Logical Domains. In *Proc. of POPL*, pages 235–246, 2008.
[11] A. Gupta, C. Popeea, and A. Rybalchenko. Predicate Abstraction and Refinement for Verifying Multi-Threaded Programs. In *Proc. of POPL*, pages 331–344, 2011.
[12] A. Gupta, C. Popeea, and A. Rybalchenko. Solving Recursion-Free Horn Clauses over LI+UIF. In *APLAS*, pages 188–203, 2011.
[13] A. Gupta and A. Rybalchenko. InvGen: An Efficient Invariant Generator. In *Proc. of CAV*, pages 634–640, 2009.
[14] N. Halbwachs and M. Peron. Discovering Properties about Arrays in Simple Programs. In *Proc. of PLDI*, pages 339–348, 2008.
[15] M. Heizmann, J. Hoenicke, and A. Podelski. Nested Interpolants. In *Proc. of POPL*, pages 471–482, 2010.
[16] T. Henzinger, T. Hottelier, and L. Kovács. Valigator: A Verification Tool with Bound and Invariant Generation. In *Proc. of LPAR*, LNCS, pages 333–342, 2008.
[17] T. Henzinger, T. Hottelier, L. Kovács, and A. Rybalchenko. Aligators for Arrays (Tool Paper). In *Proc. of LPAR*, pages 348–356, 2010.
[18] K. Hoder, A. Holzer, L. Kovács, and A. Voronkov. Vinter: A Vampire-Based Tool for Interpolation. In *Proc. of APLAS*, 2012. To appear.
[19] K. Hoder, L. Kovács, and A. Voronkov. Interpolation and Symbol Elimination in Vampire. In *Proc. of IJCAR*, pages 188–195, 2010.
[20] K. Hoder, L. Kovács, and A. Voronkov. Case Studies on Invariant Generation Using a Saturation Theorem Prover. In *Proc. of MICAI*, pages 1–15, 2011.
[21] K. Hoder, L. Kovács, and A. Voronkov. Invariant Generation in Vampire. In *Proc. of TACAS*, 2011.
[22] K. Hoder, L. Kovács, and A. Voronkov. Playing in the Grey Area of Proofs. In *Proc. of POPL*, pages 259–272, 2012.
[23] R. Jhala and K. L. McMillan. A Practical and Complete Approach to Predicate Refinement. In *Proc. of TACAS*, pages 459–473, 2006.
[24] J. Knoop, L. Kovács, and J. Zwirchmayr. Symbolic Loop Bound Computation for WCET Analysis. In *Proc. of PSI*, pages 224–239, 2011.
[25] J. Knoop, L. Kovács, and J. Zwirchmayr. r-TuBound: Loop Bounds for WCET Analysis (Tool Paper). In *Proc. of LPAR*, pages 435–444, 2012.
[26] K. Korovin. iProver - An Instantiation-based Theorem Prover for First-order Logic. In *Proc. of IJCAR*, pages 292–298, 2008.
[27] K. Korovin and A. Voronkov. Integrating Linear Arithmetic into Superposition Calculus. In *Proc. of CSL*, pages 223–237, 2007.
[28] L. Kovács. Aligator: A Mathematica Package for Invariant Generation. In *Proc. of IJCAR*, pages 275–282, 2008.
[29] L. Kovács. Reasoning Algebraically About P-Solvable Loops. In *Proc. of TACAS*, pages 249–264, 2008.
[30] L. Kovács and A. Voronkov. Finding Loop Invariants for Programs over Arrays Using a Theorem Prover. In *Proc. of FASE*, pages 470–485, 2009.
[31] L. Kovács and A. Voronkov. Interpolation and Symbol Elimination. In *Proc. of CADE*, pages 199–213, 2009.
[32] K. L. McMillan. An Interpolating Theorem Prover. In *Proc. of TACAS*, pages 16–30, 2004.
[33] K. L. McMillan. Quantified Invariant Generation Using an Interpolating Saturation Prover. In *Proc. of TACAS*, pages 413–427, 2008.
[34] A. Mine. The Octagon Abstract Domain. In *Proc. of RE*, pages 310–319, 2001.
[35] A. Rybalchenko and V. Sofronie-Stokkermans. Constraint Solving for Interpolation. In *Proc. of VMCAI*, pages 346–362, 2007.
[36] S. Srivastava and S. Gulwani. Program Verification using Templates over Predicate Abstraction. In *Proc. of PLDI*, pages 223–234, 2009.