European Research Council

ERC Starting Grant
Research proposal (Part B1)
(to be evaluated in Step 1)

Verification of Concurrent Data Structures

VerCors

Name of the Principal Investigator (PI):	Dr. Marieke Huisman
Name of the PI's host institution for the project:	Formal Methods and Tools group Faculty of Electrical Engineering, Mathematics and Computer Science University of Twente, Netherlands
Proposal full title:	Verification of Concurrent Data Structures
Proposal short name:	VerCors
Proposal duration in months:	60 months (5 years)

Proposal summary
Increasing performance demands, application complexity and explicit multi-core parallelism makes concurrency omnipresent in software applications. However, due to the complex interferences between threads in an application, concurrent software is also notoriously hard to get correct. Instead of spending large amounts of money to fix incorrect software, formal techniques are needed to reason about the behaviour of concurrent programs.

In earlier work, we developed a variant of permission-based separation logic that is particularly suited to reason about multithreaded Java programs with dynamic thread creation and termination, and reentrant locks. The VerCors project will extend expressiveness of the logic, to specify and verify concurrent data structures. The verification logic will be parameterised over the locking policy, so that a high-level specification of the behaviour of a data structure can be reused for different implementations. Thus the implementation of a concurrent data structure can be changed, without affecting correctness of the applications using it.

The logic will also be parameterised with concurrency and synchronisation primitives, so that a logic for a different programming language can be defined as an instance of the general logic. It will also be adapted to reason about programs with benign data races, *i.e.*, data races where the same value is written simultaneously by different threads. Also techniques to generate part of the specifications automatically will be developed. Finally, the logic will be adapted to a distributed setting, where data consistency between the different sites has to be maintained.

All results will be integrated in a tool set that generates and proves proof obligations automatically. It will be validated on realistic case studies.

Section 1: The Principal Investigator

1(a) Scientific Leadership Potential

Dr. Marieke Huisman is a well-established researcher in the area of program verification. She worked in particular on the following topics: verification of sequential Java programs, development and use of the Java Modeling Language (JML), compositional verification of control-flow properties for applets and other programs with procedures, verification of multithreaded Java programs, and verification of bytecode programs.

Verification of sequential Java programs During her PhD project (1996-2000), at the University of Nijmegen, Netherlands, Huisman participated in the LOOP project (Logic for Object-Oriented Programs), led by Prof. Dr. B.P.F. Jacobs. This project was the first to develop a fully tool-supported verification technique for a realistic programming language, namely Java. The semantics of sequential Java was modelled in the language of the theorem prover PVS, and the so-called LOOP compiler compiled any Java program into an appropriate PVS model. After this translation step, PVS was used to prove any desired property of the Java program.

Huisman contributed to the semantical description of Java in PVS. Further, she developed a Hoare logic for Java programs (embedded in the logic of PVS), that allowed reasoning at a higher level of abstraction. Later, this logic has been further extended to a weakest precondition calculus, enabling a higher level of automation in proofs. Further, she adapted the LOOP approach, so that the Isabelle/HOL theorem prover could also be used as target prover. In addition, Huisman led the effort of two substantial class library verification case studies: a class invariant of class Vector, and a functional correctness specification of AbstractCollection.

The LOOP project has been highly influential in the area of verification of object-oriented programs (as demonstrated by the high number of citations for the papers that emerged from this project – up to 165 – including Huisman's PhD thesis – 122 citations). For efficiency reasons, current tools directly implement a weakest precondition calculus, and generate only proof obligations. However, the justification for the correctness of these implementations is given by the formal developments of the LOOP project.

Also the European IST project VerifiCard (2001 – 2003), led by Prof. Dr. Jacobs, was inspired by the initial work on the LOOP project.

Development and use of the Java Modeling Language In 2000, after Huisman finished her PhD thesis, she moved to INRIA Sophia Antipolis, France. By that time, the Java Modeling Language (JML) emerged as a specification language for Java. Huisman has conducted large case studies using JML (80 citations in total), contributed to tool development for JML, and participated in discussions around the design of JML. She contributed in particular to the JACK tool set, and developed techniques to generate JML annotations from high-level security properties. She recently participated in a Dagstuhl seminar on semantics and tool support for JML (July 2009).

Compositional verification of control-flow properties Huisman has a long-running collaboration (since 2001) with Dr. D. Gurov, KTH, Sweden, on the compositional verification of control-flow properties of programs with procedures. This has resulted in an algorithmic verification technique, supported by a tool set, to verify control-flow properties of composed systems, by reducing these into properties of the components of the system. The work initially targeted Java Card applets (within the context of the VerifiCard project), but since then has been generalized to any components that communicate via message passing. The approach has been applied to the PACAP case study, developed by smart card producer Gemplus. The paper describing this case study received an EASST best paper award during ETAPS 2004, because of the excellent combination of theoretical results and practical applicability.

Verification of multithreaded Java programs During the last years, Huisman has extended her interest in program verification also to multithreaded programs. She studied the underlying Java Memory Model that prescribes allowed behaviours of a Java program. Further, she studied the impact of multithreading on security, and on functional verification.

To study the influence of multithreading on security, and particular on confidentiality, Huisman compared different definitions of confidentiality for multithreaded programs, found in the literature. She showed that many of these properties do not properly capture the intuitive notion of confidentiality, and she found that some definitions were essentially flawed (accepting programs with clear information leaks). In addition, she

used the self-composition approach to translate the confidentiality property that compares two program executions, into a property over a single program execution. This reformulation allows to use classical program verification techniques, and in particular model checking, to verify confidentiality of multithreaded programs. Huisman recently obtained a NWO grant (Netherlands Science Organisation) for the SlaLoM project. This grant supports a PhD student that will extend this approach to a wider class of security properties, and that will make the techniques scale to large programs.
Within the context of the European IP FET Mobius project, Huisman led the task on the verification of multithreaded Java. Within this task, she studied the Java Memory Model, in collaboration with G. Petri, INRIA Sophia Antipolis. Further, she collaborated with Dr. C. Haack, then at the University of Nijmegen, Netherlands, and Dr. C. Hurlin (formerly Huisman's PhD student, now at INRIA Bordeaux, France) on an extension of permission-based separation logic for Java. This logic allows to reason about dynamic thread creation and termination, and reentrant locks, thus capturing all essential features of multithreaded Java. The technique has been applied to several non-trivial multithreaded programming patterns, including traversal of a linked list, where each node in the list is protected by its own lock.

Verification of bytecode programs Also within the Mobius project, Huisman led a Specific Task Force on BML (Bytecode Modeling Language), the bytecode variation of JML. BML is designed in such a way that BML specifications can be stored directly in the class file. It specifies bytecode, and contains a few bytecode-specific constructs, but it is designed to be closely related with JML, to make it suitable for a high-level proof-carrying code framework. During the JML Dagstuhl meeting mentioned above, the need to store JML specifications in the class file was identified as one of the main targets for the future. BML defined a special class file format for this, to be compatible with older versions of Java. However, with the emergence of annotations in Java 5, these seem to be a better format for this. It is foreseen to start a Java Specification Request (JSR) to have support for JML in Java annotations (this initiative will be led by Dr. J. Hunt from Aicas Gmbh, Germany). The developments within this JSR will make heavy use of the developments and experiences gained around BML.

Other contributions Besides the contribution and recognition mentioned above, there are a few points more that are worth emphasising.
Huisman has an *H*-index of 16[1]. Her most cited papers are those related with her PhD project, but also recent work receives attention. For example, the recent APLAS 2008 paper on verification of reentrant lock is already cited 7 times (and most likely will be cited more in the future).
Huisman contributed significantly to the proposal of the IP FET Mobius project, and has been the INRIA site-leader until her move to the University of Twente, Netherlands. Besides the European funding and the NWO grant mentioned above, she also received two collaboration grants from INRIA (ARC ModoCop, to collaborate with several other INRIA and French research teams, and a grant to support a bilateral collaboration with KTH). Further, Huisman is a member of the Management Committee of the ESF Cost Action on Formal Verification of Object-Oriented Software.
Huisman has been editing two journal special issues, and has organised and chaired several conferences and workshops (see CV in Section 1(b) for a detailed overview). She is frequently asked to participate in program committees.
Huisman has successfully supervised two PhD students, Néstor Cataño and Clément Hurlin, and she has been involved in the supervision of several other PhD and master students (see CV). She will supervise a PhD student starting soon on the SlaLoM project mentioned above. In general, the students have been pleased with her supervisions, and their PhD projects have been completed without major problems.
At the University of Twente, Huisman has been appointed in a special tenure track position. She currently is assistant professor, but given sufficient quality of her work, she will become an associate professor within 5 years after her appointment.

Career stage Having completed her PhD on 1st of February, 2001, Huisman should be considered as a consolidator. However, she would like to bring to the attention of the panel that she has been on parental and extra-ordinary leave for approximately 20 months in total since December 2004. Huisman has two children: Christian, born 25th of March, 2005, and Charlotte, born 1st of December, 2008. Moreover, Christian has been severely ill for a long period. In addition, since December 2005 she has been working 4 days a week (80 %).

[1] Information based on Google Scholar, after manual corrections.

1(b) Curriculum Vitae Dr. Marieke Huisman
Date and place of birth: 3rd of May, 1973, Utrecht, Netherlands
Nationality: Dutch
Married to Kim Sunesen, mother of Christian (born 2005) and Charlotte (born 2008)

Affiliation
Formal Methods and Tools Group
Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
Netherlands
tel. +31 53 489 46 62
Marieke.Huisman@ewi.utwente.nl

Academic education

2000	PhD Thesis: Java program verification in higher order logic with PVS and Isabelle University of Nijmegen, Netherlands Supervisors: Prof. Dr. B.P.F. Jacobs, Prof. Dr. H.P. Barendregt, Dr.ir. H. Meijer
1996	Master Thesis: The calculation of a polytypic parser Utrecht University, Netherlands Supervisor: Prof. L.G.L.T. Meertens *Cum laude*

Professional record

2008 –	Assistant professor on tenure track position, to become associate professor within 5 years, Formal Methods and Tools (FMT) Group, University of Twente, Netherlands
2001 – 2008	*Chargée de Recherche* (researcher), first in the Lemme project, later in the Everest project, led by Dr. G. Barthe, at INRIA Sophia Antipolis
2000 – 2001	Post doc. INRIA Sophia Antipolis
1996 – 2000	PhD student at the University of Nijmegen, Netherlands

Key figures

Number of publications in refereed journals:	3
Number of publications in refereed proceedings:	25
Number of citations:	1051
H-index:	16
Number of PhD students supervised:	4
Number of workshops and conferences co-chaired	6
Number of edited special issues journals	2

Supervision of PhD students
- Ngo Minh Tri, to start on NWO funded project SlaLoM, February 2010 – 2013. *Security by Logic for Multithreaded Applications.*
- Gustavo Petri (via alfa project LerNet, 2007 – 2008, after Huisman left INRIA, Gustavo continued under the supervision of Dr. G. Boudol). *Specification and verification of the Java Memory Model for multi-threaded applications.*
- Clément Hurlin (2006 – 2009). *Specification and Verification of Multithreaded Object-Oriented Programs with Separation*
- Néstor Cataño (2001 – 2004). *Formal methods for Java programs.*
- Kerry Trentelman (PhD student ANU, Australia, during her 6 month visit to INRIA). *Aspects of Java Program Verification.*

Supervision of master students
- Henri-Charles Blondeel (Supelec, France and KTH, Sweden, 6 months, 2007). *Security by logic: characterising non-interference in temporal logic.*
- Gustavo Petri (6 months, 2006). *Formalisation of the Java Memory Model.*
- Alejandro Tamalet (5 months, 2006). *Annotation Generation.*
- Pratik Worah (IIT Kharagpur, 3 months, 2004). *Logical characterisation of observational determinism.*

- Mariela Pavlova (co-supervision, Univ. of Paris 7, 6 months, 2004). *Generation of JML specifications.*
- Néstor Cataño (6 months, 2001). *JML's modifiable clause: semantics, verification and application.*

PhD committees
- Joachim van den Berg (2nd of July, 2009). *Reasoning about Java programs in PVS using JML.*

Administrative duties
- Member of management committee ESF COST Action on Formal Verification of Object-Oriented Software.
- FMT contact person for the CTIT SRO ISTRICE (Integrated Security and Privacy in a Networked World).
- Vice-leader of the Everest project team at INRIA (until 2008).
- Task leader of the task on Multithreading in the IP FET Mobius project. Coordinator of the Specific Task Force that defines BML (Bytecode Modeling Language). Contributed to submission and coordination of this project (see http://mobius.inria.fr).
- Participant in French national research project ParSec (2007 – 2010, see http://moscova.inria.fr/~zappa/projects/parsec/parsec.html).
- Site-coordinator for the French national research project Geccoo (2003 – 2006), see http:// geccoo.lri.fr).
- Coordinator of collaborative project with KTH, Sweden on compositional verification of control-flow security properties for applets (2005).
- Coordination of the French national research project ModoCop (Model Checking of Concurrent Object Oriented Programs, 2001 – 2003)
- Participant in IST project Verificard, 2001 – 2003, co-organiser of several project meetings.

Organisational and editorial duties
- Co-organisation of the SAVCBS workshop (Specification and Verification of Component-Based Systems) at ESEC/FSE 2009.
- Co-editor of special issue of the Journal of Object Technology for FTfJP and IWACO 2008.
- Co-organisation of the FTfJP workshop (Formal Techniques for Java-like Programs) at ECOOP 2008.
- Co-organisation of the VAMP workshop (Verification and Analysis of Multithreaded Java-like Programs) at Concur 2007.
- Member of the jury for the Isabelle Attali best technical paper award at e-Smart 2005 and 2006.
- Co-organisation of the Cassis workshop (Construction and Analysis of Safe, Secure and Interoperable Smart devices) in Marseilles (2004), and Nice (2005). Proceedings published as LNCS 3362 and 3956, Springer.
- Co-editor of special issue of the Journal of Logic and Algebraic Programming on Formal Methods for Smart Cards, 2004.
- Co-organiser of VeriSafe workshop (joint workshop of VerifiCard and SecSafe projects), Nice, 2002.
- Co-organiser of final workshop of ModoCop project, Grenoble, 2003.
- Co-organiser of Lemme project seminar, INRIA Sophia Antipolis, 2000 – 2001.
- Co-organiser of Computing Science Institute seminar, University of Nijmegen, Netherlands, 1998 – 2000.
- Co-organiser of ProTaGoNist (Proof Tools Group Netherlands), and 2nd Dutch Proof Tools Day, University of Nijmegen, Netherlands, 1997.

Program committees
- SAVCBS 2009 (chair), 2007, 2006 – ESEC/FSE workshop
- FTfJP 2008 (chair), 2007, 2004 – ECOOP workshop
- VAMP 2007 (co-chair) – Concur workshop
- Bytecode 2007 (co-chair), 2005 – ETAPS workshop
- VSTTE 2006 – FLoC workshop
- .NET Technologies 2006, 2004
- Cassis 2005 (co-chair), 2004 (co-chair)

Funding ID

Current grant: NWO (Netherlands Science Organisation) support for one PhD student (4 years) to work on Security by Logic for Multithreaded Applications (see http://fmt.cs.utwente.nl/projects/SlaLoM/). This project involves 20 % of Huisman's time. Because of the focus on multithreaded applications, it will match well with the research goals described in the current proposal.

1(c) Early Achievement-Track-Record

Publications

All papers are available at http://wwwhome.ewi.utwente.nl/~marieke/papers.html. All papers marked with a (*) are without Huisman's PhD supervisor as co-author. Number of citations based on information from Google Scholar and manually corrected.

Journal papers

- (*) D. Gurov, M. Huisman, and C. Sprenger. *Compositional verification of sequential programs with procedures.* Information and Computation, 206:840-868, 2008. (6 citations)
- C. Breunesse, N. Cataño, M. Huisman, and B. Jacobs. *Formal methods for smart cards: an experience report.* Science of Computer Programming, 55(1-3):53-80, 2005. (35 citations)
- M. Huisman, B. Jacobs, and J. van den Berg. *A Case Study in Class Library Verification: Java's Vector Class.* Software Tools for Technology Transfer, 3/3:332-352, 2001. (63 citations)

Refereed conference papers

- (*) J. Chrzaszcz, M. Huisman, A. Schubert. *BML and related tools.* In Software Technologies Concertation on Formal Methods for Components and Objects (FMCO 2008). LNCS 5751, pp. 278-297, Springer, 2009.
- (*) M. Huisman and A. Tamalet. A Formal Connection between Security Automata and JML Annotations. In Fundamental Approaches to Software Engineering (FASE 2009). LNCS 5503, pp. 340-354, Springer, 2009. (2 citations)
- (*) D. Gurov and M. Huisman. Reducing Behavioural to Structural Properties of Programs with Procedures. In Proceedings of VMCAI 2009, LNCS 5403, pp. 136-150, Springer, 2009. (4 citations).
- (*) C. Haack, M. Huisman, and C. Hurlin. *Reasoning about Java's Reentrant Locks*. In 6th Asian Symposium on Programming Languages and Systems (APLAS 2008). LNCS 5356, pp. 171-187, Springer, 2008 (7 citations)
- (*) M. Huisman, I. Aktug, and D. Gurov. *Program models for compositional verification.* In ICFEM 2008, LNCS 5256, pp. 147-166, Springer, 2008. (4 citations)
- (*) L. Burdy, M. Huisman, and M. Pavlova. *Preliminary design of BML: A behavioral interface specification language for Java bytecode.* In Fundamental Approaches to Software Engineering (FASE 2007), LNCS 4422, pp. 215-229. Springer, 2007. (18 citations)
- (*) G. Barthe, L. Burdy, J. Charles, B. Grégoire, M. Huisman, J.-L. Lanet, M. Pavlova, and A. Requet. *JACK: a tool for validation of security and behaviour of Java applications.* In Formal Methods for Components and Objects, LNCS 4709, pp. 152-174. Springer, 2007. (25 citations)
- (*) M. Huisman, P. Worah, and K. Sunesen. A *temporal logic characterization of observational determinism.* In 19th IEEE Computer Security Foundations Workshop. IEEE Computer Society, 2006. (16 citations)
- (*) D. Gurov and M. Huisman. *Interface abstraction for compositional verification.* In SEFM'05, pp. 414-423. IEEE Computer Society, 2005. (3 citations)
- (*) M. Huisman and K. Trentelman. Factorising *temporal specifications*. In M. Atkinson and F. Dehne, editors, Computing: A Theory Symposium (CATS 2005), volume 41, pp. 87-96. ACSC, 2005. (1 citation)
- (*) M. Huisman, D. Gurov, C. Sprenger, and G. Chugunov. *Checking absence of illicit applet interactions: a case study*. In M. Wermelinger and T. Margaria, editors, Fundamental Approaches to Software Engineering (FASE 2004) LNCS 2984, pp. 84-98. Springer, 2004. (9 citations)
- (*) M. Pavlova, G. Barthe, L. Burdy, M. Huisman, and J.-L. Lanet. *Enforcing high level security properties for applets.* In J.-J. Quisquater, P. Paradinas, Y. Deswarte, and A.A. El Kalam, editors, Cardis'04, pp. 1-16. Kluwer, 2004. (35 citations)
- (*) C. Sprenger, D. Gurov, and M. Huisman. *Compositional verification for secure loading of smart card applets.* In C. Heitmeyer and J.-P. Talpin, editors, Memocode 2004, pp. 211-222. IEEE, 2004. (13 citations)
- (*) N. Cataño and M. Huisman. *Chase: a Static Checker for JML's Assignable Clause.* In L.D. Zuck, P.C. Attie, A. Cortesi, and S. Mukhopadhyay, editors, Verification, Model Checking and Abstract Interpretation (VMCAI '03), LNCS 2575, pp. 26-40. Springer, 2003. (36 citations)

- (*) N. Cataño and M. Huisman. *Formal specification and static checking of Gemplus's electronic purse using ESC/Java.* In L.-H. Eriksson and P.A. Lindsay, editors, Formal Methods Europe (FME'02), LNCS 2391, pp. 272-289. Springer, 2002. (45 citations)
- (*) G. Barthe, D. Gurov, and M. Huisman. *Compositional verification of secure applet interactions.* In R.-D. Kutsche and H. Weber, editors, Fundamental Approaches to Software Engineering (FASE'02), LNCS 2306, pp. 15-32. Springer, 2002. (17 citations)
- (*) M. Huisman. *Verification of Java's AbstractCollection class: a case study*. In E. Boiten and B. Möller, editors, Mathematics of Program Construction (MPC'02), LNCS 2386, pp. 175-194. Springer, 2002. (10 citations)
- (*) K. Trentelman and M. Huisman. *Extending JML Specifications with Temporal Logic*. In H. Kirchner and C. Ringeissen, editors, Algebraic Methodology and Software Technology (AMAST'02), LNCS 2422, pp. 334-348. Springer, 2002. (35 citations)
- (*) G. Barthe, G. Dufay, M. Huisman, and S. Sousa. *Jakarta: a toolset for reasoning about JavaCard.* In I. Attali and T. Jensen, editors, Smart Card Programming and Security (E-Smart'01), LNCS 2140, pp. 2-18. Springer, 2001. (16 citations)
- M. Huisman and B. Jacobs. *Inheritance in higher order logic: Modeling and reasoning*. In J. Harrison and M. Aagaard, editors, Theorem Proving in Higher Order Logics: 13th International Conference (TPHOLs 2000), LNCS 1869, pp. 301-319. Springer, 2000. (32 citations)
- M. Huisman and B. Jacobs. *Java program verification via a Hoare logic with abrupt termination.* In T. Maibaum, editor, Fundamental Approaches to Software Engineering (FASE 2000), LNCS 1783, pp. 284-303. Springer, 2000. (139 citations)
- J. van den Berg, M. Huisman, B. Jacobs, and E. Poll. *A type-theoretic memory model for verification of sequential Java programs.* In D. Bert, C. Choppy, and P.D. Mosses, editors, Recent Trends in Algebraic Development Techniques, LNCS 1827, pp. 1-21. Springer, 2000. (69 citations)
- B. Jacobs, J. van den Berg, M. Huisman, M. van Berkum, U. Hensel, and H. Tews. *Reasoning about classes in Java (preliminary report).* In Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '98), pp. 329-340. ACM Press, 1998. (165 citations)
- (*) W.O.D. Griffioen and M. Huisman. A *comparison of PVS and Isabelle/HOL*. In J. Grundy and M. Newey, editors, Theorem Proving in Higher Order Logics: 11th International Conference (TPHOLs '98), LNCS 1479, pp. 123-142, 1998. (35 citations)
- U. Hensel, M. Huisman, B. Jacobs, and H. Tews. *Reasoning about classes in object-oriented languages: Logical models and tools.* In C. Hankin, editor, Proceedings of European Symposium on Programming (ESOP '98), LNCS 1381, pp. 105-121. Springer, 1998. (16 citations)

Theses

- M. Huisman. Reasoning *about Java programs in higher order logic using PVS and Isabelle*. PhD thesis, Computing Science Institute, University of Nijmegen, 2001. (122 citations)
- M. Huisman. *The calculation of a polytypic parser*. Master Thesis, Utrecht University, 1996. (6 citations)

Awards

- EASST Best Paper Award at ETAPS 2004 for: M. Huisman, D. Gurov, C. Sprenger, and G. Chugunov. Checking absence of illicit applet interactions: a case study. In M. Wermelinger and T. Margaria, editors, Fundamental Approaches to Software Engineering (FASE 2004), LNCS 2984, pp. 84-98. Springer, 2004.
- Freye stipendium, travel grant from the University of Nijmegen, Netherlands, for female PhD student to support long term visits abroad, used for a 3 months visit to the Automated Reasoning Group, led by Mike Gordon and Larry Paulson, at Cambridge University, UK (1999).

1d: Extended Synopsis of the VerCors proposal

Verification of Concurrent Data Structures (VerCors)

1. Motivation

With the increasing demands on efficiency and computing power, the concurrent programming paradigm has become omnipresent. Hardware is not becoming significantly faster anymore; instead modern computer architectures have multiple cores. Thus, to make full use of the computing power of such machines, applications have to distribute the computing tasks over multiple threads. Multi-core model checking is a typical example: by distributing the model checking tasks over different cores, larger systems can be handled in the same time. In addition, multiple threads are also used to increase responsiveness of applications. For example, in a mobile phone, the main thread should be non-blocking, so that it can always react upon an incoming phone call. Therefore, any other activity will be spawned as a separate thread. Also, in many applications, graphical user interfaces are programmed as separate threads, so that the (often costly) operations on the graphical user interface do not impact the performance of the program itself.

Unfortunately, writing concurrent software is error-prone. Since executions of threads can be interleaved in many different ways, the total number of executions in a program is quickly too large for a human to grasp. Thus, for a programmer it is easy to overlook a special situation that might occur, and to introduce an error in the program. As a result, not only the production but also the maintenance of software constitutes a huge cost. Large amounts of money are being spent for coping with erroneous software, and there are not many fields where research has a similar potential for savings.

One of the most fruitful techniques for improving software productivity and software quality is catching large numbers of errors at compile-time, *i.e.*, before running the software. This project does exactly this, and does this where this is most needed, namely for concurrent software. In particular, the project will deliver a collection of verified implementations of concurrent data structures. In addition, a tool set will be developed that allows one to prove formally that applications that use these data structures are correct. Moreover, the tool set will automatically generate specifications to help the prover wherever possible. The logic that underlies the verification technique will be parametric over locking policies – including lock-free implementations – and concurrency and synchronisation primitives. This allows to instantiate the verification technique for a different programming language easily. Finally, the verification technique will also be generalised to a distributed setting.

It is expected that after completion of this project, the verification technique and tool set will be usable for advanced developers of concurrent software, and that it will actually help them to increase the quality of their applications. Currently, formal Hoare logic-based specification and verification techniques for sequential applications, such as JML, are reaching a state where they are usable in industry for realistic applications. For concurrent applications, such techniques do not yet exist. However, the recent emergence of separation logic – an extension of Hoare logic – makes reasoning about concurrent programs manageable, because it allows thread-modular verification. Therefore, we believe that now all ingredients are available to develop a verification technique for concurrent applications that is both sound, and practically usable. We expect that after completion of the VerCors project, industrials can use similar tools as exist for JML to reason about concurrent applications.

2. State-of-the-art in Program Verification of Concurrent Applications

The verification of functional specifications of concurrent applications has long been a challenge. Owicki and Gries [12] were the first to develop a program logic-based verification method for parallel programs. Later, Jones adapted this to the compositional rely-guarantee method [9]. However, with both approaches, verification does not scale to realistic programs.

O'Hearn was the first to show that separation logic is suited to reason about concurrent programs [11]. However, O'Hearn's approach is too restrictive, because it enforces that two different threads cannot *read* the same variable simultaneously, while this situation occurs frequently in concurrent programs, and is harmless. Extending separation logic with permissions, as introduced by Boyland [2], solves this. Permissions are used to specify whether a variable is read-only or can be changed. In each program state, variables are associated with permissions. A full permission (1) gives the right to change a variable, while a fractional permission (any value between 0 and 1) only gives the right to read a variable. Thus, permission-based separation logic can be used to reason about concurrent programs where multiple reads to the same

location are allowed. If the total number of permissions to access a certain location is never more than 1, this guarantees that only reads can happen simultaneously. A thread can only write to a location if it has the full permission to do this, and therefore all other threads have permission 0 to access this location, *i.e.*, they cannot even read it.

Independently of each other, several groups applied this idea to more realistic concurrent programming languages. Gotsman *et al.* [3] and Hobor *et al.* [6] showed how permission-based separation logic can be used to reason about languages with single-entry locks, dynamic thread creation and POSIX threads, *i.e.*, a C-like language. Haack, Huisman and Hurlin [4,5,7] applied permission-based separation logic to reason about a Java-like language with reentrant locks, garbage collection, inheritance and subclassing, and Java-based concurrency primitives. Leino *et al.* [10] developed Chalice: a programming language and verification tool. Chalice does not reason using separation logic; instead, specifications contain explicit access predicates, and these specifications are translated into first-order logic before generating proof obligations. Their language contains single-entry locks, dynamic thread creation and termination.

Typical properties that are verified using these approaches are for example absence of data races or deadlock. However, these approaches are not yet suited to specify and verify functional specifications of applications, describing their precise behaviour. Moreover, they are tailored to one specific programming language.

3. Application Domain: Concurrent Data Structures

To make full use of the power of multi-core architectures, programs use efficient concurrent data structures, where multiple threads can store and retrieve data simultaneously. Typical examples are, among many others, large central data bases that can be accessed from different sites, concurrent hash tables used for (multi-core) model checking, and request buffers for thread pools. Since concurrent data structures are at the core of many concurrent applications, the VerCors project will target them as dedicated application domain.

Moreover, many different implementations of such concurrent data structures are possible, that differ for example in the locking policy that is being used. A different locking policy can greatly impact the performance of the application that uses the concurrent data structure, but it should not change its behaviour. Special attention will be given to data structures that do not use any locks at all, *i.e.*, so-called *lock-free* or *wait-free* data structures.

Independent of the policy that is chosen to control access to the data structure, the data structure should behave as expected. In fact, its specification can be split into two parts: the first part describes the functional behaviour of the data structure, the second part is related with the concurrency aspects. This first part should correspond to the specification of a sequential implementation of the data structure, the second part differs for each locking policy. If the correctness of an application only depends on the first part of the specification, this allows changing the implementation, and to choose the one with the most efficient locking policy.

4. Objectives

Goal of the VerCors project is to develop specification and verification techniques for concurrent data structures, written in a wide range of programming languages, using different concurrency paradigms. Particular points of interest will be: different locking policies; lock-free data structures; variations in concurrency and synchronisation techniques; expressiveness and readability of specification language; generation of permission annotations; distributed data structures; and tool support.

5. Methodology

Permission-based Separation Logic Basis of the work conducted in the VerCors project will be the permission-based separation logic developed by Haack, Huisman and Hurlin [4,5,7]. This logic allows to reason about a Java-like language with dynamic thread creation and termination and reentrant locks in a thread-modular way. The logic contains the standard operators from separation logic, *i.e.*, the points-to predicate, the separating conjunction (the star operator), and the magic wand (or separating implication). In addition, each pointer is associated with a permission that specifies whether the owner of the pointer can either read or update the location that is being pointed to. Permissions are a value in the domain [0, 1]. Soundness of the verification system ensures a global correctness criterion, namely that the total number of permissions in the system pointing to a particular location never exceeds 1. If a thread has a pointer with permission 1, this means that this thread is the only thread that can access this location, and thus the location can safely be modified. If a thread only has a fractional permission, *i.e.*, a permission that is less than 1, this means that other threads might also be accessing the same location simultaneously, and thus the value stored there can only be read, and not updated. Notice that a program can only be verified with this logic, if it does not contain any data races (and of course, if it respects the method specifications).

Tool Support, Specification Language and Automated Verification All techniques that will be developed within the project will be supported by appropriate tools, making use of modern IDE (such as Eclipse) to provide an attractive and effective user interface. To this end, we will join efforts with the developers of the JML plug-in for Eclipse. In addition, we will combine permission-based separation logic with JML, resulting in an expressive and readable specification language that is suited for concurrent programs. Concretely this means that the expression grammar of JML will be extended, and that separation logic's abstract predicates will be unified with JML model methods.
Further, it is important that proof obligations can be verified automatically. If we would translate the separation logic assertions into first-order logic, this could result in expressions with nested quantifications, which are an impediment to automation [1]. Instead, we plan to combine especially tailored proof-theoretical decision procedures for logical entailment of a restricted subset of separation logic [1], with the use of a linear logic constraint solver [8]. We will improve those methods to cover the full subset of separation logic that occurs in the proof obligations. Further, we also need support to reason about the absence of aliasing, because of reentrant locks. In our verification method, each lock is associated with a resource invariant that is obtained upon acquiring the lock. However, if a lock is acquired reentrantly, the associated resource invariant should not be obtained – otherwise resources are duplicated, which causes the system to be unsound. Thus, a thread can only safely obtain a resource invariant for a lock l if l is not held by the thread yet – and thus it has to be proven that no alias of l is in the set of all the locks held by the thread. In earlier work, we have shown how this problem could be handled by combining separation logic assertions with ownership types. We will explore how this can be done in a more systematic way, so that the proof obligations about absence of aliasing can be discharged automatically.
Finally, all theoretical developments that are described below will be integrated in the tool set and validated on realistic case studies.

Concurrent Data Structures with Different Locking Policies A collection of concurrent data structures will be specified and verified. Initially, an implementation with a fine-grained locking policy will be specified, and then the specification will be adapted for an implementation with the same behaviour, but a different locking policy. Eventually, this will allow splitting the specifications into two parts: the first part describing the abstract behaviour of the data structure, the second part being related with the concurrency aspects and the particular locking policy of the implementation. This means that a single specification can be verified for several implementations that vary in the degree of locking. Moreover, any application that can be verified on the basis of the abstract behaviour only is correct for any implementation that varies only in the locking policy.

Variations in Concurrency and Synchronisation Techniques Concurrent programming languages all use different primitives to model concurrency and synchronisation. Verification techniques for concurrent programs that have been developed so far have studied different thread mechanisms, but they have mainly concentrated on simple locks (possibly reentrant) as a synchronisation primitive. However, in actual applications, a much larger variety of synchronisation techniques is used: reader-writer locks, latches, barriers, futures etc. A classification of synchronisation techniques will be developed, and for each class of techniques, appropriate verification techniques will be studied. In particular, the APIs implementing these techniques have to be formally specified (and verified) in such a way that they can be used as any other synchronisation technique in verifications. In addition, appropriate automated verification procedures will be developed for the proof obligations that result from these synchronisation primitives. Finally, the verification logic will be parameterised with concurrency and synchronisation primitives, so that it can be easily instantiated for different programming languages.

Generation of Permission Annotations A large burden for program verification is that it is time-consuming and error-prone to write the specifications. In particular, to ensure that verification succeeds, often many trivial details have to be specified. For sequential programs, experiments have been conducted to generate part of the specifications, based on the actual implementation. For concurrent programs, a similar approach will be used to generate specifications that state that some code fragment only reads a variable, while other code fragments update it. In addition, alias analysis can be performed to improve the precision of the generated annotations. If several aliases to a certain memory location exist, this can give an indication that permissions should be split. Also, we will generate possible monitor invariants by deriving from the program code what state is actually protected by a lock, *and* which permissions are obtained by acquiring the lock.
For the generation of annotations, we plan to use an approach that is based on a combination of weakest precondition calculus and static analysis. The first technique can tell us under which conditions we can

guarantee that a method will terminate normally, the second will provide us with additional assumptions that we can make on the program and data structure, which can help us to improve the precision of our results.

Lock-Free Data Structures Since they do not constrain concurrency, lock-free data structures can positively influence the performance of an application, but they also require special care, because data might be corrupted, duplicated or removed. We will study how the specifications for concurrent data structures with locks can be adapted to the lock-free case. The specifications will make explicit what guarantees can be provided, or under which additional conditions a lock-free data structure functions correctly. For example, many algorithms still function correctly if only benign data races occur, or under an occasional data loss. To extend the verification logic to handle benign data races, specific variables can be marked explicitly as permission-free. Two flavours of permission-free variables can exist: those allowing benign data races only, and those allowing arbitrary data races. A program is verified if all permissions to access a variable never exceed 1, except for these permission-free variables. For an arbitrary permission-free variable, no check is applied at all; for a benign permission-free variable, verification of the program ensures that always the same value is written when the data race occurs. To understand what behaviours a verified program allows, memory models such as the Java Memory Model will have to be studied, and in particular, the claims that the effects of race conditions are local and that programs with benign data races are still sequentially consistent will have to be proven formally.

Distributed Data Structures In modern architectures, data structures often are not only used concurrently, but also distributed over different nodes in the system. This requires that the distributed parts of the data structure exchange information, and ensure some global consistency properties, similar to how cache-coherence is ensured. Special protocols are used to ensure consistency, and the verification logic will be used to prove correctness of the protocol.
The verification logic will be extended to distributed data structures, by providing rules to reason about data that is moved from one site to another. Standard APIs that implement this will be specified and verified. Further, the underlying heap model has to be extended to consider that data can be distributed over different sites, and even might be duplicated on different sites. The rules in the logic will be adapted and proven sound w.r.t. this extended distributed heap model.

6. Resources and Distribution of Tasks

The VerCors project will run for five years. The total budget that is requested for the project is 1.306.500 Euros. The PI will be involved in the project for 50 % of her time during the full duration of the project. Further, funding is asked for two PhD students (PhD projects run for four years in the Netherlands), two post docs (for three years each), and a scientific programmer.
The first PhD student will work on specification and verification of concurrent data structures with different locking policies. The second PhD student will work on generalising the verification techniques to different classes of concurrency and synchronisation techniques, and on the correctness of lock-free data structures. The first post doc will work on the development of an appropriate specification language, and on the automated verification of proof obligations, by developing or extending appropriate solvers for separation logic, and developing ownership type-based techniques to prove the absence of aliasing automatically. The second post doc will study the automatic generation of specifications, and an extension of the developed techniques to distributed data structures.
The last focus point of the project is tool support. Both PhD students and post docs are expected to implement their results. In addition, the project will also fund a scientific programmer, who will ensure that the different developments are combined into a single tool, provide an effective user interface and guarantee maintenance of the tool. To ensure that the outcome of the project is a tool that actually is more than an academic prototype, and that it can be used by external parties, we think it is essential to have a professional developer funded by the project.
The distribution of work is summarised in Table 1. The PI is not explicitly mentioned, as she will be involved in the initiation and supervision of all these tasks. Dark coloured fields mean that a person is involved in the task; a light coloured field means that a person will contribute to discussions and provide input where possible. The numbers indicate in which years of the project the task is expected to take place.

7. Summary and Conclusions

Goal of the VerCors project is to develop efficient techniques for the specification and verification of concurrent data structures, written in a wide range of programming languages, using different concurrency paradigms. This will result in an efficient static verification technique for concurrent applications that will

Tasks	PhD 1	PhD 2	Post doc1	Post doc2	Sc. Prog.
Tool support	**1-4**	**2-5**	**1-3**	**3-5**	**1-5**
Integration of separation logic and JML			**1**		**1**
Automated verification techniques for proof obligations			**1-3**		
Concurrent data structures with different locking policies	**1-4**		**1-3**		
Variations in concurrency and synchronisation techniques		**2,3**		**3**	
Generation of permission annotations				**3,4**	**3,4**
Lock-free data structures		**4,5**		**4,5**	
Distributed data structures				**4,5**	
Case studies	**3**	**4,5**	**3**	**4,5**	

Table 1: Distribution of Tasks

significantly improve software quality and reduce maintenance costs. Key project results include: a large collection of fully specified and verified implementations of concurrent data structures; a verification technique that is parametric over locking policy and synchronisation and concurrency primitives, and that also allows to reason about lock-free programs with (benign) data races; an extension of the verification technique to a distributed setting; and full tool support, including a readable, expressive specification language, automated generation of specifications, and automated verification of proof obligations.

The omnipresence of concurrent applications makes the results of the VerCors project highly relevant with a huge potential for cost savings in application development and maintenance. Recent theoretical advances, such as the development of separation logic and JML, have led to new insights, which make verification of concurrent applications feasible in practice.

The PI has the proven experience, knowledge, and leadership to head this project. She has a wide experience with the use of permission-based separation logic for concurrent applications; she knows formalisations of memory models underlying concurrent applications; she has experience with tool support for program verification; she is well-established in the area of program verification, the JML specification language, and separation logic, and she has good managerial qualities.

Bibliography

1. J. Berdine, C. Calcagno, and P.W. O'Hearn. *Smallfoot: Modular automatic assertion checking with separation logic*. In Formal Methods for Components and Objects, LNCS 4111, pp. 115-137, Springer, 2005.
2. J. Boyland. *Checking interference with fractional permissions.* In R. Cousot, eds., Static Analysis Symposium, LNCS 2694, pp. 55-72. Springer, 2003.
3. A. Gotsman, J. Berdine, B. Cook, N. Rinetzky, and M. Sagiv. *Local reasoning for storable locks and threads*. In Z. Shao, eds., Asian Programming Languages and Systems Symposium, LNCS 4807, pp. 19-37. Springer, 2007.
4. C. Haack, M. Huisman, and C. Hurlin. *Reasoning about Java's reentrant locks*. In G. Ramalingam, eds., Asian Programming Languages and Systems Symposium, LNCS 5356, pp. 171-187. Springer, 2008.
5. C. Haack and C. Hurlin. Separation logic *contracts for a Java-like language with fork/join.* In Algebraic Methodology and Software Technology, LNCS 5140, pp. 199-215. Springer, 2008.
6. A. Hobor, A. Appel, and F.Z. Nardelli. *Oracle semantics for concurrent separation logic*. In European Symposium on Programming (ESOP), LNCS 4960, pp. 353-367. Springer, 2008.
7. C. Hurlin. *Specification and Verification of Multithreaded Object-Oriented Programs with Separation Logic*. PhD thesis, Université de Nice Sophia Antipolis, 2009.
8. L. Jia and D. Walker. *Ilc: A foundation for automated reasoning about pointer programs.* In European Symposium on Programming (ESOP), LNCS 3924, pp. 131-145, Springer, 2006.
9. C.B. Jones. *Tentative steps toward a development method for interfering programs.* ACM Transactions on Programming Languages and Systems, 5(4):596-619, 1983.
10. K.R.M. Leino, P. Müller, and J. Smans. *Verification of concurrent programs with Chalice*. In Lecture notes of FOSAD, LNCS 5705. Springer, 2009.
11. P.W. O'Hearn. *Resources, concurrency and local reasoning*. Theoretical Computer Science, 375(1(3):271-307, 2007
12. S. Owicki and D. Gries. An *axiomatic proof technique for parallel programs*. Acta Informatica Journal, 6:319-340, 1975.