Our proposal falls in the domain of static program analysis, which is inherently linked to informatics. More specifically, our proposal corresponds to the intersection of the following domains of the W&T5 expert panel: theoretical computer science, parallel and distributed computing, programming languages and techniques, and software engineering.
Summary in layman’s terms
As concurrency becomes prevalent, static analyses supporting contemporary concurrency models are necessary. Existing analysis designs for concurrency suffer from limitations stemming from the challenges that concurrency brings. First, concurrent programs are highly non-deterministic, leading to poor scalability of existing designs. Second, dynamic process creation is a crucial feature of modern concurrent programs that only few designs account for. Third, no single unified design supports the various concurrency models, and multiple concurrency models lack static analysis support. Finally, although concurrency models are often combined, no static analysis design accounts for this. We developed and formalized two analysis design methods providing a first step towards a solution to these four problems, and applied them to two common concurrency models. These general designs should be amenable to supporting additional concurrency models. We propose to extend our work along three axes. First, we will investigate extending our designs to other contemporary concurrency models (atomics, channels, and STM) for which static analysis support is currently lacking. The insights gained in this manner will enable refining our designs. Second, we will investigate supporting combinations of concurrency models in these designs, as no existing analysis supports such combinations. Finally, we will improve the precision of each analysis by addressing limitations identified during our research.

RESEARCH

Main host institution  Vrije Universiteit Brussel
Head of the research unit  De Meuter Wolfgang ()
Additional host institution(s)
Additional head of the research unit  Jonckers Viviane ()
Gonzalez Boix Elisa ()
De Roover Coen ()
Collaboration  not applicable

EXTRA DATA

Supervised dissertations
Noah Van Es, “Performant Scheme interpreter in asm.js”, BA Thesis 2015-2016
Funding applied for elsewhere or already available
None

Previous FWO fellowships
None

Previous research stays
None

Planned research stays
Plans for a research stay have not yet been concretized, but the applicant is eager to collaborate internationally through 3-month stays (e.g., with the founders of the AAM method to abstract interpretation) after the completion of his doctoral dissertation.

Scientific awards
No
I obtained my engineering degree at the Université Libre de Bruxelles (ULB) with the greatest distinction, for which I completed a Master’s thesis at the Software Languages Lab (SOFT) of the Vrije Universiteit Brussel (VUB) on static analysis of concurrent programs. Even though I had not studied at this university, the positive experience of the Master’s thesis made me decide to join the VUB to pursue a PhD. In the first months there, I managed to publish about my master’s thesis at the 2015 PPDP conference, which confirmed my professional interest in research and in static program analysis in particular.

My attraction to work that attempts to unify related but dissimilar concepts naturally led me to study abstract interpretation in detail, a general theory of abstraction applicable to many different problems in computer science. It also led me to formulate two formal designs for precise, sound and scalable static analyses of concurrent programs rooted in abstract interpretation. As I find demonstrating the practicalities of these designs as important as the formal proofs of their properties, I also put actual implementations of these designs to the test by instantiating them from a general framework that I gradually evolved to support each analysis. My research method of combining a strong formalization with a thoroughly evaluated practical implementation is reflected in the work I have done for my PhD. There I applied my analysis designs to two different concurrency models, proved crucial theoretical properties of the resulting analyses using mechanized formal proofs, and demonstrated their more practical characteristics through a thorough empirical evaluation. The work resulted in several high-quality publications, including a paper presented at the prestigious ECOOP 2017 international conference and two articles still under review at top-tier journals.

The past four years immersed at SOFT have enabled me to study existing static analyses and their underlying theories, to master formal proofs and methods, and to propose, formalize, prove, and evaluate new analysis designs. While the ideas developed during my PhD have already been applied to two concurrency models, I strongly believe that they merit further study in the context of other concurrency models and combinations thereof. As I am in the process of finishing the text for my PhD dissertation (promoted by Prof. dr. Coen De Roover and by Prof. dr. Wolfgang De Meuter), a post-doctoral scholarship would at this moment provide a natural continuation enabling the further development of these ideas and of my academic career. There is no doubt in my mind that pursuing more research on the topic of static analysis of concurrent programs will prove successful, in terms of publication, personal development, and advances to the state of the art.

**Career breaks**

None
How this project fits in the research activities of the research group

The Software Languages Lab (SOFT) of the Vrije Universiteit Brussel will host the proposed research. This group originates from the 2009 merger of the Programming Technology Lab with the Systems and Software Engineering Lab, each founded in 1987. Over the years, SOFT has grown to an internationally-renowned group with expertise in programming language and software engineering research. Over 40 PhDs and a multitude of high-end publications in international conference proceedings and journals have resulted.

SOFT is composed of multiple focus groups, and this project is situated at the confluence of two of these focus groups. The first group is led by Prof. Coen De Roover and consists of about 10 researchers with expertise in static analysis and program transformation in general, and in abstract interpretation of dynamic languages such as Scheme and JavaScript in particular. A second group is led by Prof. Wolfgang De Meuter and consists of another 10 researchers that focus on language design and implementation for multi- and many-core computing platforms. The applicant, already a member of the first focus group since the start of the PhD research, will remain in this focus group but collaborate closely with the second focus group.

National and international context

The Software Languages Lab has several ongoing international collaborations of an informal nature (e.g., joint organization of conferences and workshops, frequent invitation to PhD juries,...), and is coordinator of or partner in several collaborative research projects that are tangentially related to this proposal.

The FWO/FNRS-funded EOS project “Automated Assistance for Developing Software in Ecosystems of the Future” (SECO-ASSIST), for instance, will investigate ecosystem-scale software analytics from January 2018 onwards. Its start coincides with the start of the Innoviris-funded TeamUp project INTiMALS which investigates applications of mining software repositories in the context of software renovation. Finally, the VLAIO-funded “Secure Coordination of Rich Internet Application Tiers” (Tearless) project investigates applications of program analysis in the context of web applications until the end of 2019. The complete list of ongoing research projects can be consulted on its website.

The applicant is eager to strengthen the existing national and international collaborations, as well as start new collaborations within the research community on program analysis and concurrency. He has already been a member of the artifact evaluation committee of the community’s OOPSLA conference since 2017. He considers several 3-months research stays abroad as an ideal means towards this end. The strengthened collaborations could lead to a number of project applications in the future, for example as part of the EU-funded projects of the 7th Framework Programme (FP7) in the areas of Software Engineering, Internet of Services and Cloud Computing.
In the table below questions are listed on the ethical aspects of your research proposal.

If you mark a ‘yes’ for the question, it follows that

- **For the questions marked with *:** the applicant is legally or on the basis of institutional regulations obliged to ask for an ethical advice at the competent ethics committee of the host institution; please do take into account that even when there is no obligation with regard to the research itself, for the publication of the results a positive advise still can prove to be necessary.

  If you have answered questions with a * positively, you must submit your proposal to the ethics committee **as soon as your application has been approved for funding.** Your fellowship can only start when this clearance has been formally given. Only if the advice relates to a work package that is planned for a later stage of the fellowship, it may be submitted just before the start of that part of the research. Please keep in mind that the advisory procedure can take some time and that therefore you should submit your proposal to the ethics committee **well in time.**

- **For the questions that are not marked:** the applicant and the evaluation panel are invited to reflect on the issue and take, if necessary, the necessary precautionary measures.

  You find more on the FWO policy and procedure concerning ethical issues and on legal and other documents on the [FWO web page dedicated to that topic](http://www.fwo.be).

I confirm that none of the issues below apply to my proposal.  
True

I hereby confirm having taken note that an ethical clearance is needed for the start of my project. I will thus ensure submission of my proposal to the research ethics committee of my host institution.

Please specify which ethics committee(s) deal(s)/will deal with your application.

In case you will submit your proposal to the committee only before the start of work package(s) (WP) that are concerned:

<table>
<thead>
<tr>
<th>Number/description of WP(s)</th>
<th>Starting date of WP(s)</th>
</tr>
</thead>
</table>

Applicant: Quentin Stiévenart | Application number: 12X2719N
1. Human Embryos/Foetuses

ETHICS ADVICE RELATED TO THESE QUESTIONS SHOULD ALWAYS BE REQUESTED BEFORE THE START OF
THE RESEARCH PROJECT AS A WHOLE AND ALSO REQUIRE AN EXAMINATION BY THE FEDERAL
COMMISSION FOR EMBRYOS

Does your research involve Human Embryonic Stem Cells (hESCs)?
- Will the hESCs be directly derived from embryos within this project?
- Are the hESCs previously established cell lines?

Does your research involve the use of human embryos?

Does your research involve the use of human foetal tissues / cells?

2. Humans

Does your research involve human participants?
- Are they volunteers for social or human sciences research?
- Are they persons unable to give informed consent?
- Are they vulnerable individuals or groups?
- Are they children/minors?
- Are they patients?
- Are they healthy volunteers for medical studies?

Does your research involve physical interventions on the study participants?
- Does it involve invasive techniques?
- Does it involve collection of biological samples?

3. Human Cells/Tissues

Does your research involve human cells or tissues (other than from Human Embryos/Foetuses, i.e. section 1)?
- Are they obtained from commercial sources?
- Do they originate from another laboratory/institution/biobank?
- Were they produced or collected by you from previous research activities?
- Are they produced or collected by you as part of this project?

4. Personal Data

Does your research involve personal data collection and/or processing? (¹)
- Does it involve the collection and/or processing of sensitive personal data?
- Does it involve collecting/processing of genetic information/data?
- Does it involve tracking or observation of participants?

Does your research involve further processing of previously collected personal data (‘secondary use’)?

---

Applicant: Quentin Stiévenart | Application number: 12X2719N
5. Animals

Does your research involve research procedures to live non-human vertebrate animals (incl. independently feeding larval forms, foetal forms of mammals in the last trimester of their normal development and cephalopods, and also forms in earlier stages if the experiments have consequences in later stages)?*  

- Are they vertebrates or live cephalopods?  
  - N/A  
- Are they non-human primates? (*)  
  - N/A  
- Are they genetically modified animals?  
  - N/A  
- Are they cloned farm animals?  
  - N/A  
- Are they endangered species?  
  - N/A  

6. International Collaboration

Do you plan to use local resources (e.g. animal and/or human tissue samples, genetic material, live animals, human remains, materials of historical value, endangered fauna or flora samples, etc.)?  

- No  

Do you plan to import/export any material from/to other countries?  

- No  

Name of country/ies:  

If your research involves low and/or lower middle income countries, are benefits-sharing measures foreseen?  

- N/A  

Could the situation in the country put the individuals taking part in the research at risk?  

- N/A  

7. Environment & Health and Safety

Does your research involve the use of elements that may cause harm to the environment, to animals or plants?  

- No  

Does your research deal with endangered fauna and/or flora and/or protected areas?  

- No  

Does your research involve the use of elements that may cause harm to humans, including research staff?  

- No  

8. Dual Use

Does your research have the potential for military applications?  

- No  

9. Misuse

Does your research have the potential for malevolent/criminal/terrorist abuse?  

- No  

10. Other Ethics Issues

Are there any other ethics issues that should be taken into consideration? Please specify.
For these issues the Belgian commission on privacy protection (Commissie voor de bescherming van de persoonlijke levenssfeer) has to be consulted. You cannot consult the commission directly, but always first contact the research coordination of your host institution.

In this case you already have to submit your proposal to the ethics committee in the application phase.
PROJECT OUTLINE

Indicate the state of the art.

A static analysis reasons about the possible behaviors of a program without executing it. This is useful for a number of applications such as program comprehension, bug detection, and automated proofs. Static analysis of concurrent programs is a challenging undertaking for a number of reasons, but its necessity is growing as concurrent and parallel programs become more and more prominent. We review the major challenges faced when devising a static analysis for concurrent programs.

Poor scalability

Concurrent systems are highly non-deterministic in their execution, as multiple processes are executed concurrently and their executions may interleave in an exponential number of possible ways. This is known as the state explosion problem [63]. Mitigations for this problem have been studied in the context of model checking, leading to techniques such as partial-order reduction [18, 21]. Although these techniques reduce the number of interleavings an analysis has to explore, they do not solve the explosion problem itself as the number of interleavings to explore remains exponential. Also, while such techniques are sound for defect detection (any detected error is a true error) and can prove correctness for finite systems, they fall short when analyzing concurrent systems that may exhibit infinite behavior. Abstract interpretation [9] is a well-studied technique able to reason about infinite systems, but partial-order reduction techniques are not compatible with abstract interpretation because they do not support the cyclic state spaces present in abstract interpretation. A more adequate solution to the state explosion problem in static analysis consists of modularizing an analysis [10] by analyzing parts of the program in isolation. Interleavings of different processes are not explicitly represented in the state of a modular analysis but are still soundly over-approximated. Although not prone to the state explosion problem, only a few and rather limited modular analyses have been designed explicitly for concurrent programs [17, 28, 37, 38].

Lack of support for dynamic process creation

Modern concurrent programs exhibit highly dynamic behavior such as dynamic process creation, a challenging feature for static analysis [27] that so far has been addressed by only a few analysis techniques. Existing techniques featuring static partial order reduction or a modular analysis approach do not support dynamic process creation, which limits them to analyzing programs with a fixed set of processes. While these techniques may scale well and verify complex concurrent systems [41], they fall short when considering modern concurrent systems composed of processes that are created at runtime. Existing work supporting dynamic systems is meager, an exception being the Soter tool [14] that fails to scale beyond synthetic benchmarks due to its non-modularity [58].

Support for a limited number of concurrency models

The model of threads with shared memory is a well-established concurrency model available in widely used programming languages. Support for this model in static analysis exists in the form of model checking and abstract interpretation. However, these analyses either scale well without support for dynamic process creation [17, 28, 38] or are subject to scalability issues despite attempts at mitigating the inherent state explosion problem [18, 19, 21]. The actor model [2, 24] is a concurrent model growing in popularity over the past years due to the advent of microservices [23, 43]. However, only few static analyses support the actor model [14, 47, 56] and they face the same limitations. Analyses for other concurrency models are few and far between. A number of static analyses have been developed for channel concurrency [26], but they are limited to programs with
fixed number of processes [35, 37, 44, 52]. Static analysis of software transactional memory (STM) [48] has been developed primarily in the context of compiler optimization [1], with a focus on providing safer STM models [50] instead of bringing static analysis to the existing model. Static analysis of atomics [25] has been studied in the form of abstract interpretation [68] but fails to scale due to the explicit modeling of all execution interleavings [55].

No support for combinations of concurrency models
Programs written in concurrent languages such as Java, Scala, Clojure, Rust, or Go often combine multiple concurrency models. For example, Scala developers mix the actor model with other models in 80% of the applications studied in [62], and around half of the 684 developers interviewed in [20] use both shared-memory and message-passing concurrency. This motivates the need for static analyses supporting combinations of concurrency models, in particular because mixing different concurrency models breaks assumptions made by each model, thereby introducing new types of bugs [22, 32, 33, 51]. Recent work studies safe combinations of different concurrency models [12, 13, 32, 42, 59, 61, 71], or develops debugging tool support for combinations of concurrency models [34]. So far, however, no static analysis targets combinations of concurrency models.

Towards analysis design methods for scalable analysis of concurrency models
Contemporary concurrent programs are highly non-deterministic and rely on dynamic process creation, two features that are not taken into account simultaneously by existing static analyses. We demonstrated the scalability issues that arise when applying abstract interpretation to a minimal concurrent language with dynamic process creation and atomics in our PPDP 2015 paper [55], motivating the need to address these issues in static analysis of concurrent programs. We have developed two generic analysis design methods to achieve sound and scalable static analysis of concurrent programs exhibiting dynamic process creation. First, in our ECOOP 2017 paper [56], we adapted macro-stepping semantics to abstract interpretation and demonstrated that it improves the scalability of the analysis without impacting its precision. Second, we have addressed the state explosion through modularity, inspired by the work of Cousot and Cousot [10]. Our second analysis design method results in a scalable analysis that analyzes each concurrent process in isolation. The latest results are presented in two papers that are under review at top-tier journals [53, 58]. We expect the review of the first paper on the 2nd of February, but have already made both papers available to the reviewers of this proposal1. Our work represents a solid first step towards scalable analysis of concurrent programs with support for dynamic process creation, but we believe there is still ample opportunity for improving the precision of the resulting analyses, while also transposing them to other commonly used concurrency models and combinations thereof.

Describe the objectives of the research
In our previous work, we devised two analysis design methods for obtaining scalable program analyses. Both methods rely on annotating a transition relation with communication effects that describe concurrent operations performed by the program under analysis. Our first design method mitigates the state explosion problem by using the communication effects information to soundly ignore certain process interleavings. This results in larger steps of the transition relation being analyzed for single processes, which is why we call an analysis resulting from this method a macro-stepping analysis [56]. Being amenable to abstract interpretation and supporting dynamic process creation, such an analysis overcomes the limitations of typical partial-order reduction techniques. Our second analysis design method results in a modular analysis [53, 58] which analyzes

each process in isolation while collecting the generated communication effects without directly applying them to the global analysis state. Once a process has been fully analyzed, the impact of the collected effects are propagated and analysis of other processes proceeds in the same fashion. The result is a sound scalable analysis, unaffected by the state space explosion problem as interleavings of the execution are not explicitly explored by this analysis but are still explicitly accounted for.

Existing static analyses supporting concurrency generally target either thread or actor models of concurrency, and do not address scalability and dynamic process creation at the same time. We therefore propose to extend our existing work [53, 56, 58] to achieve precise and scalable analysis of actors and threads with support for dynamic process creation, to other prevalent concurrency models. Our hypothesis is that our analysis design methods will give rise to sound, precise, and scalable static analyses for other concurrency models without having to abandon their support for dynamic process creation. In addition, we will investigate support for combinations of concurrency models in our analysis design, as such combinations are prevalent among contemporary programming languages. We will refine our analysis design methods in the process, and improve the precision of the analyses by addressing the sources of imprecisions identified in our previous work.

O1: Instantiating our analysis design method for prevalent concurrency models

Our first objective is to design precise, sound, and scalable static analyses for the most prevalent concurrency models [59] not addressed in our previous work: atomics [25], channels [26], and STM [48]. This will be achieved by applying our analysis design methods [53, 56, 58], which incorporate macro-stepping and modularity in the analysis design. Atomics are present in modern languages featuring concurrency such as Java, Scala, C and Clojure, but no scalable static analysis supports this concurrency feature. Static analysis support for channels exists [35, 44, 52], even in the form of modular analysis [37], but these analyses are limited to programs with a fixed number of processes, while channel abstractions are available in many languages that support dynamic process creation (e.g., Clojure, Java, Scala, Rust, and Go). There is finally no proper static analysis support for STM with a focus on proving program properties or performing program comprehension. The application of our analysis design methods to these concurrency models will therefore result in new precise and scalable static analyses for each of these models which is amenable to implementations for real-world languages. In addition to the analyses resulting from the application of our design methods, the endeavour will lead to the gradual refinement of the methods themselves. More concretely, we will first develop a calculus-specified in PLT Redex—for each concurrency model. Next, we will design the analysis for each calculus according to our design methods, formalizing it in Coq to prove termination and soundness, as well as implementing it in our Scala-AM static analysis framework [57] to evaluate its precision, performance, and scalability. Insights gained in this manner will be incorporated in our design methods and re-evaluated on the other analyses resulting from its application.

O2: Instantiating our analysis design methods for combinations of concurrency models

Concurrency models are often combined in practice and research has started to account for this fact by developing new models [59, 61] and debugging tools that support such combinations [34]. The focus of this objective is therefore to support combinations of concurrency models in static analysis. The combinations we focus on are driven by the concurrency models that are already being combined in practice. First, languages such as Java, Scala and Clojure allow the combination of atomics, threads with shared memory, and actors, but without any enforcement mechanism to prevent incorrect uses, e.g., race conditions between accesses to shared state among two actors. Second, languages featuring channel-based concurrency such as Rust, Go, and soon Scala support
both synchronous and asynchronous channels. The study of synchronous channels is part of the first objective, while asynchronous channels can be seen as a variant of actors, which we already studied in previous work [53, 56, 58]. Although this combination of channel concurrency is used in practice [5], no existing static analysis supports it. Finally, STM and actors have been combined [30, 60, 69], but this combination currently lacks static analysis support to prevent potential issues such as messages being sent more than once when sent within a transaction that is rolled back [60]. Focusing our research on these three combinations provides an immediately useful and solid foundation for analyzing programs that employ different concurrency models. Because our design methods are described in a generic formulation, they are applicable to multiple concurrency models at the same time, while existing methods tend to be linked with a specific concurrency model and not amenable to support multiple models. Insights gained from the application of our design methods to combinations of concurrency models will be incorporated into the methods.

O3: Improving the precision of static analyses resulting from our design method

Whereas the first two objectives focus on developing analysis support for a wider range of concurrency models, these analyses may become imprecise on larger programs. Also, soundness is a crucial property to preserve in static analysis, although this property is often deliberately ignored by commercial static analyzers in order to improve precision [7]. Our third objective therefore focuses on realizing analyses for concurrent programs with high precision without sacrificing soundness. While automated analysis techniques have sometimes been criticized for their imprecision [15], analyses of sequential programs that deal with real-world programs of millions of lines of code while still producing a low number of false positives exist [8]. However, in order to obtain a high precision, care needs to be taken when designing the analysis. While for sequential programs there has been focus on identification of sources of imprecision [3, 70] and design of precise abstract domains [40], investigating precision of concurrent programs analysis remains meagre. Existing techniques for concurrent programs tend to sacrifice soundness to improve precision [6, 45, 46], and only a few focus on the precision impact of sound techniques [39].

We envision three orthogonal approaches to improve the precision for concurrent programs. The first is to apply context sensitivity [49] to process creation, resulting in what we call process sensitivity, for improved precision by accounting for the program’s execution history at process creation. Second, the analyses resulting from our modularity-inspired design [53, 58] currently lack order information—although we have shown in other work that order information may have a crucial impact on precision in static analysis of certain properties of concurrent programs [56]. We therefore aim to incorporate order information into our analysis design methods, taking inspiration from existing preliminary work [37], which lacks support for dynamic process creation. Finally, as both the modularity-based design method and the macro-stepping analysis design method rely on the same core semantics, we plan on combining modular analysis with macro-stepping analysis in the same design method. This allows a more fine-grained trade-off between the scalability provided by modular analysis and the precision provided by macro-stepping analysis. Each of these precision improvement techniques will be proven sound through our Coq formalization.

Describe the methodology of your research.

We will use the analysis design methods developed in our previous research as a base for this research. Both methods share the same foundation of relying on a transition relation annotated with communication effects, describing the concurrent operations, on which we perform abstract interpretation [9] in the AAM style [68]. Our first method reduces the number of interleavings to explore and results in a macro-stepping analysis [56], mitigating the state explosion problem. Our
second method solves the state explosion problem by analyzing each process in isolation and collecting the communication effects that may be performed, in order to discover new processes and new interactions to analyze. The analysis resulting from this design method is called a modular analysis and scales well [53, 58].

We applied both our analysis design methods to the actor model and to a shared-memory multi-threaded model. We will similarly apply both methods to other concurrency models and combinations thereof, incorporating new insights gained from these applications into our design methods. We will first formalize each concurrency model (or combination of models) in a calculus representative of the main feature of that model. We will use the PLT Redex semantics engineering toolkit [16] for this purpose to increase confidence in the correctness of our models [29]. We will then apply both our methods to these calculi, mechanizing their formalization in the Coq proof assistant [4]. This formalization will be used to prove crucial theoretical properties such as soundness and termination. We will also implement all the resulting analyses in Scala-AM [54, 57], a static analysis framework and that has already proven its use for experimenting with various static analysis techniques [11, 64–67]. This will provide a high-quality reproducible evaluation of our research: theoretical properties of the model are formally proven through executable mechanized proofs, and practical properties are evaluated experimentally on sets of benchmarks.

Provide a work plan, i.e. the different work packages and a detailed timetable.

As represented in the Gantt chart below, we divide our work into three iterations over the three objectives we outlined, each iteration focusing on a concurrency model: the first iteration concerns atomics, the second iteration concerns channels, and the final iteration concerns STM. Each work package aims at a new contribution to the state of the art, and therefore a scientific publication. Each iteration is estimated to take around one year and is divided into three work packages following the same structure. In the first work package of each iteration (WP1, WP4, WP7) we develop a calculus for the concurrency model under consideration, with the help of the PLT Redex semantics engineering tool [16]. We develop a number of programs in an extended version of this calculus, featuring programs inspired from common problems in the concurrency literature and exhibiting dynamic process creation, in order to obtain a benchmark suite representative of the concurrency model under consideration. We then apply our analysis design methods to the developed calculus, resulting in a macro-stepping analysis and a modular analysis. We formalize the resulting analyses in the Coq proof assistant [4] to prove soundness and termination, and implement the analyses in our Scala-AM framework [57]. We use our benchmarks to assess the scalability and precision of each analysis. Finally, we retroactively improve our analysis design methods based on the insights gained from applying them to a new concurrency model. In the second work package of each iteration (WP2, WP5, WP8), we perform the same process for the combination of concurrency models under consideration. Because our methods are described in a generic way, they are suited to support more than one concurrency model at a time, and we will again incorporate insights from applying them into the design. In a third work package (WP3, WP6, WP9), we investigate a sound precision improvement technique that the analyses developed in this iteration will profit from.
WP1: Static analysis of atomics
The first work package consists in revisiting our older work on static analysis of atomics [55] by applying our more recent analysis design methods [53, 56, 58] to a calculus featuring multi-threaded concurrency with atomic variables that model shared memory. This application would share similarities to the application of our design methods to multi-threaded programs with mutable references [53]. The resulting macro-stepping analysis and modular analysis will be shown scalable through an implementation and proven sound through a mechanized formalization. We will also measure their precision and incorporate new insights gained from this work into our analysis design methods.

WP2: Analyzing combinations of atomics, shared memory, and actors
Actors tend to be mixed with other concurrency models such as atomics and shared memory [62], and new models accounting for this combination are being developed [12, 31, 42]. Therefore, in this work package we first develop a new calculus combining atomics, shared mutable references and actors. We then apply our analysis design methods to this calculus. We have previously investigated mutable references [53] and actors [53, 56, 58], and atomics are the focus of WP1. The starting point of this work package is to combine these different analyses. This is achieved by applying our analysis design methods, and the resulting analyses will then be proven sound and evaluated in terms of precision and scalability. Such analyses are capable of targeting modern languages such as Scala and Java that allow the combination of these concurrency models.

WP3: Improving precision through process-sensitivity
Context-sensitivity [49] is used in static analysis of functional languages to improve precision with respect to function calls. To improve precision in analysis of concurrent languages with respect to process creations, this work package results in the concept and study of process-sensitivity, a novel technique intended to improve the precision of an analysis based on information about the previously created processes, which would benefit the analyses designed in the first two work packages and in the subsequent work packages.

WP4: Static analysis of channels
The second iteration focuses on channels à la CSP [26]. Analyses for channels exist, but none of them support dynamic process creation in a scalable manner. In this work package we develop a calculus supporting channels as used in modern concurrent languages, to which we apply our analysis design methods, resulting in the first scalable and sound static analyses supporting channels and dynamic process creation. We again evaluate the analyses in terms of precision and scalability on a number of benchmarks, and propagate insights gained in this work package into our analysis design methods.

WP5: Analyzing combinations of channels and actors
Channels and actors both fall under the umbrella of communicating processes. The main difference between the two models is that while actors communicate asynchronously, channels provide a synchronous means of communication. Due to their similarities, they can often be combined, and in modern languages such as Go, Rust or Scala, both asynchronous and synchronous channels can be used. This motivates the need for an analysis supporting both models at the same time, which we design in this work package. We therefore design a calculus integrating a combination of channels and actors, to which we apply our analysis design methods. We evaluate the resulting analyses in terms of precision, scalability and soundness and incorporate insights gained from this work package into our design methods.
WP6: An analysis design method that combines modularity and macro-stepping
Applying our modularity-based analysis design method results in a scalable technique, but comes at the cost of precision when compared to a macro-stepping analysis. We will devise an analysis design method that combines both modularity and macro-stepping by applying a modular analysis to the global concurrent system, which in turn will launch a macro-stepping analysis on individual processes or groups of processes that require more precision. This will mainly benefit the precision of analyses of channel concurrency. Instead of analyzing each process in isolation, it becomes possible to group multiple processes sharing a channel within a macro-stepping analysis, thereby increasing the precision with respect to a given channel while preserving overall modularity and thus scalability. This work package therefore aims at providing a more flexible trade-off between the scalability provided by modular analysis and the precision provided by macro-stepping analysis.

WP7: Static analysis of STM
The third and final iteration focuses on STM. Only a handful of static analyses have targeted STM, although STM is present in modern languages such as Haskell, Clojure, and Scala. In this work package, we develop a calculus integrating STM, on which we apply our analysis design methods, thereby developing the first full-fledged general analyses for this type of concurrency. The analyses will be evaluated in terms of soundness, precision and scalability.

WP8: Analyzing combinations of STM and actors
As combinations of STM with actors have been proposed in languages such as Scala and Clojure, this work package is dedicated to the development of an analysis that supports the use of both models simultaneously. This is done by developing a calculus to express combinations of these models and by applying our analysis design methods on this calculus. The resulting analyses will be evaluated in terms of soundness, precision and scalability, and can then be used to ensure the safety of the operations performed under multiple concurrency models, because assumptions made by one of the models (e.g., the serializability of transactions or the isolation of actors) may be broken when combined with other models [60].

WP9: Incorporating order in modular analysis
Our modularity-based analysis design method incurs a cost in terms of precision that is mostly due to the loss of order in the inferred communication effects for each analyzed process. However, order is important in STM as operations made within a transaction must happen according to a specific order. Preliminary work that supports order in modular analysis has appeared recently [36], although limited to a setting with two predefined processes. In this work package, we will use this work as inspiration to incorporate the concept of order in our modularity-based analysis design method. Analyses resulting from applying this new method would in theory benefit from precision improvement, and we will evaluate the practical impact of this improvement on precision, as well as the cost in terms of scalability.

PUB: Dissemination of research results
Each work package aims at advancing the state of the art in concurrent program analysis and should therefore result in a publication, hence we reserve a work package for the dissemination of our research results. This research is rooted in static analysis, program verification, abstract interpretation and programming languages, and we therefore target high-quality venues such as ECOOP, PLDI, ESOP, SPLASH, ICSE, CAV, SAS, ICFP, and POPL, as well as a number of top journals.
Enumerate the bibliographical references that are relevant for your research proposal.

157–168.
Indicate below whether you think the results of the proposed research will be suitable to be communicated to a non-expert audience and how you would undertake such communication.

Even though communication to a non-expert audience is a challenging undertaking, given that our work at the very least requires some knowledge about static analysis and concurrent programming, we believe there exist several appropriate channels to communicate the results of our research to a broader audience. First, we can maintain a blog where we explain our experiences and findings at a high-level, as well as provide background material on static analysis, abstract interpretation, and concurrency models. We can also describe the use of tools or methods that we use in our research, such as the Coq proof assistant, the PLT Redex semantics engineering toolkit, and the Scala-AM static analysis frameworks. In addition we will make all our source code publicly available. Scala-AM is already open-sourced on GitHub, and by continuing work on this repository we will make our own contributions available for developers to build upon. We will also release, showcase and publicize tools build on top of our research that bring static analysis support to modern concurrent languages.
Dear

concerning your application for an FWO postdoctoral fellowship:

please take into account that you have to obtain your PhD degree before 02/06/2018 to be eligible.

Thanks in advance.

with kind regards
your FWO administrator
Ilse Raveel
Dear applicant,

In your application for an FWO mandate you have proposed ten external referees appointed at a university, research institution or research entity of another type of organization.

Unfortunately insufficient referees have responded to the FWO-invitation. Therefore I would like to ask you to provide us with five additional referees at the latest by Wednesday 21 March 2018. Please send the contact details (name, institute, e-mail address) by e-mail to this address.

The regulations concerning the peer review can be found on our website: http://www.fwo.be/en/the-fwo/organisation/fwo-expertpanels/regulations-fwo-internal-and-external-peer-review/

Thank you
Kind regards,
Ilse Raveel
Account administration
Dear Ilse,

I'm sending you this mail to notify the FWO that I have successfully obtained my PhD degree today, as I passed the public defense successfully.

Let me know if you need any other information regarding this.

Best regards,
Quentin
<table>
<thead>
<tr>
<th>Surname</th>
<th>First name</th>
<th>Email</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rajan</td>
<td>Hridesh</td>
<td><a href="mailto:hridesh@iastate.edu">hridesh@iastate.edu</a></td>
<td>Iowa State University</td>
</tr>
<tr>
<td>Hu</td>
<td>Zhenjiang</td>
<td><a href="mailto:hu@nii.ac.jp">hu@nii.ac.jp</a></td>
<td>National Institute of Informatics</td>
</tr>
<tr>
<td>Haller</td>
<td>Philipp</td>
<td><a href="mailto:phaller@kth.se">phaller@kth.se</a></td>
<td>KTH Royal Institute of Technology</td>
</tr>
<tr>
<td>Sen</td>
<td>Koushik</td>
<td><a href="mailto:ksen@cs.berkeley.edu">ksen@cs.berkeley.edu</a></td>
<td>University of California</td>
</tr>
<tr>
<td>Visser</td>
<td>Eelco</td>
<td><a href="mailto:e.visser@tudelft.nl">e.visser@tudelft.nl</a></td>
<td>TU Delft</td>
</tr>
<tr>
<td>Van Horn</td>
<td>David</td>
<td><a href="mailto:dvanhorn@cs.umd.edu">dvanhorn@cs.umd.edu</a></td>
<td>University of Maryland</td>
</tr>
<tr>
<td>Hendren</td>
<td>Laurie</td>
<td><a href="mailto:hendren@cs.mcgill.ca">hendren@cs.mcgill.ca</a></td>
<td>McGill University</td>
</tr>
<tr>
<td>Schüpp</td>
<td>Sibylle</td>
<td><a href="mailto:schupp@tuhh.de">schupp@tuhh.de</a></td>
<td>Hamburg University of Technology</td>
</tr>
<tr>
<td>Lämmel</td>
<td>Ralf</td>
<td><a href="mailto:laemmel@uni-koblenz.de">laemmel@uni-koblenz.de</a></td>
<td>University of Koblenz-Landau</td>
</tr>
<tr>
<td>Christakis</td>
<td>Maria</td>
<td><a href="mailto:maria@mpi-sws.org">maria@mpi-sws.org</a></td>
<td>Max Planck Institute for Software Systems</td>
</tr>
<tr>
<td>Vinju</td>
<td>Jurgen</td>
<td><a href="mailto:Jurgen.Vinju@cwi.nl">Jurgen.Vinju@cwi.nl</a></td>
<td>Centrum Wiskunde &amp; Informatica Amsterdam</td>
</tr>
<tr>
<td>Might</td>
<td>Matthew</td>
<td><a href="mailto:might@uab.edu">might@uab.edu</a></td>
<td>University of Alabama at Birmingham</td>
</tr>
<tr>
<td>Møller</td>
<td>Anders</td>
<td><a href="mailto:amoeller@cs.au.dk">amoeller@cs.au.dk</a></td>
<td>Aarhus University</td>
</tr>
<tr>
<td>Bodden</td>
<td>Eric</td>
<td><a href="mailto:eric.bodden@uni-paderborn.de">eric.bodden@uni-paderborn.de</a></td>
<td>Universität Paderborn</td>
</tr>
<tr>
<td>Flatt</td>
<td>Matthew</td>
<td><a href="mailto:mflatt@cs.utah.edu">mflatt@cs.utah.edu</a></td>
<td>University of Utah</td>
</tr>
</tbody>
</table>
PERSONAL DETAILS

Stiévenart Quentin

Date of birth 4 April 1991
Country of birth Belgium
Nationality Belgium
Gender M
E-mail quentin.stievenart@gmail.com

PAST AND CURRENT STUDIES

<table>
<thead>
<tr>
<th>Degree</th>
<th>Diploma</th>
<th>Country</th>
<th>University</th>
<th>Graduation date</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ph.D.</td>
<td>Computerwetenschappen</td>
<td>Belgium</td>
<td>Vrije Universiteit Brussel</td>
<td>25 May 2018</td>
<td>no entry</td>
</tr>
<tr>
<td>Master</td>
<td>Ingenieurswetenschappen: computerwetenschappen</td>
<td>Belgium</td>
<td>Université Libre de Bruxelles</td>
<td>1 July 2014</td>
<td>degree with greatest distinction</td>
</tr>
<tr>
<td>Bachelor</td>
<td>Ingenieurswetenschappen</td>
<td>Belgium</td>
<td>Université Libre de Bruxelles</td>
<td>1 July 2012</td>
<td>degree with distinction</td>
</tr>
</tbody>
</table>

DISCIPLINES

Software engineering
Programming languages and technologies
Language design, constructs and features
Language processors (compilers, interpreters, virtual machines)
Coding tools and techniques, testing and debugging

CAREER

Overview of positions NOT related to FWO and connected to a receiving university/organization
Overview of FWO-related positions

Position | Organization | Country | Start date | End date
---|---|---|---|---
PhD student (100.00%) | Vrije Universiteit Brussel | Belgium | 1 August 2014 | 1 June 2018

**SCIENTIFIC PUBLICATIONS**

<table>
<thead>
<tr>
<th>Articles</th>
<th>Year of publication</th>
</tr>
</thead>
</table>

**Other publications**

(C1) Papers in proceedings of scientific conferences, that do not belong to any of the previous categories (full articles, no abstracts). In chronological order, starting with the most recent item, with full bibliographic description.

<table>
<thead>
<tr>
<th>Articles</th>
<th>Year of publication</th>
</tr>
</thead>
</table>

(C4) All other publications or items of scientific output which are relevant to the application and cannot be included in any of the previous categories.

<table>
<thead>
<tr>
<th>Articles</th>
<th>Year of publication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>2017</td>
<td>Quentin Stiévenart, Jens Nicolay, Wolfgang De Meuter, Coen De Roover.</td>
</tr>
<tr>
<td></td>
<td>Poster at the conference on Systems, Programming, Languages and</td>
</tr>
<tr>
<td></td>
<td>Applications: Software for Humanity (SPLASH 2016).</td>
</tr>
<tr>
<td>2016</td>
<td>Quentin Stevenart, Jens Nicolay, Coen De Roover and Wolfgang De</td>
</tr>
<tr>
<td></td>
<td>Meuter. Poster at the 29th European Conference on Object Oriented</td>
</tr>
<tr>
<td>2015</td>
<td>Quentin Stievenart, Jens Nicolay, Wolfgang De Meuter and Coen De</td>
</tr>
<tr>
<td>2014</td>
<td>Quentin Stiévenart, Jens Nicolay, Coen De Roover, Wolfgang De</td>
</tr>
<tr>
<td></td>
<td>Meuter. Presentation abstract; presented at the Workshop on Software</td>
</tr>
</tbody>
</table>