ERC Advanced Grant 2015

Research Proposal (Part B2)

Principal investigator: Helmut Veith
Host institution: TU Wien (Technische Universität Wien), Austria
Proposal full title: Harnessing Model Checking for Distributed Algorithms
Proposal short name: HYDRA

Section 2: The Project Proposal

(a) State of the art and Objectives

(a.1) State of the art

Distributed algorithms constitute an important and mature field of computer science with thousands of research publications summarized in textbooks such as [14, 97, 107]. **For a discussion of their practical impact and the importance of verification, we refer to Part B1 of this proposal.** In this section, we focus on formal verification and formal modeling with an emphasis on parameterized model checking.

Formal Models for Distributed Algorithms

Since the mid 80ies, there has been a growing awareness about the need for formal models in distributed algorithms. Among multiple approaches [34, 117, 72, 114, 73], the following two pioneered by Nancy Lynch and Leslie Lamport have gained most traction (cf. also Part B1, p.3 for more discussion)

(a) TLA+[1] is a very expressive logic which entrusts the designer with the full power of set theory, i.e., the possibility to use arbitrarily complex mathematics to specify an algorithm. Thus, the design process in TLA+ is more guided by mathematical insight and skill than by language principles: TLA+ is powerful, very generic but quite unstructured, untyped, and hard to analyze. PlusCal is a macro language on top of TLA+ which supports the specification and verification of concurrent programs in a more structured und restricted way [92]. *In principle*, TLA+ can describe algorithms different from concurrent programs, but the specification of e.g. a communication topology requires the user to define graphs from first principles in terms of set theory; there is no specific support from the language. TLA+ has been used in industry by Intel [19], Compaq [80], and most recently Amazon [104] who is a collaboration partner for HYDRA. The most important TLA+ tool from our perspective is the model checker TLC which is applicable to moderately sized finite-state non-parameterized restrictions of the systems; *due to lack of parametrization, TLC is a debugging rather than a verification tool.* Current work on TLA+ is focusing on an interface to theorem proving [49, 102].

(b) Input/Output automata (IOA)[2] and their extension to Timed Input/Output Automata (TIOA) [82] are geared to a more specific domain – message passing algorithms – for which they provide a natural input language. In a series of theses [124, 23, 122, 68], the IOA project developed a simulator, a connection to the (discontinued) Larch theorem prover, a code generator, a translator to TLA+ (to use model checking in TLC), and an experimental interface to the Daikon invariant detector [57]. IOA found important use as a theoretical model, but to the best of our knowledge, the IOA toolset is not actively supported and maintained.

TLA+ and IOA were mainly used as formal frameworks for conducting proofs by hand or using proof assistants. For instance, Lamport's solution to the Byzantine Generals Problem specified in TLA+ was proven by hand [93]. Several distributed algorithms written in TLA+, IOA, or pseudo code were proven correct with proof assistants: a majority voting and a replicated storage algorithms specified in IOA and proven with Larch [23]; DiskPaxos specified in TLA+ and proven with Isabelle/HOL [75]; synchronous Byzantine agreement proven with PVS [96]; Last Voting Consensus in the heard-of model proven with Isabelle [35].

Taking a fresh look at the seminal papers for IOA and TLA+, we find that their arguments give strong support to the HYDRA project. Although both approaches attract significant interest and contributions, we were not able to locate significant numbers of example specifications in TLA+ and IOA. Thus, the potential of both approaches has not been fully realized. The HYDRA project will follow up on their agenda, and provide powerful

[1]**"The TLA Home Page"** `http://research.microsoft.com/en-us/um/people/lamport/tla/tla.html`
[2]**"IOA Language and Toolset"** `http://groups.csail.mit.edu/tds/ioa/`

tools which exploit and extend the state of the art in model checking for the analysis of distributed algorithms. In addition to verification tools, we will provide an online repository for models of distributed algorithms, and conduct verification competitions to attract interest in the repository, cf. p. 11. We believe that these additions to the tool chain will add and revive interest in the existing specification languages for distributed algorithms.[3]

In discussions with Mark Tuttle (Principal Engineer at Intel, coinventor of IOA, and user of TLA+) during preparation of the HYDRA proposal, he explicitly supported our plan to advance the state of the art.

Non-parameterized Model checking

A model checker is an efficient algorithm to decide if a piece of program code or a hardware description has a specified property, for instance, absence of deadlocks, non-violation of assertions (safety), loop termination (liveness), API conformance etc.; if the property is violated, the model checker outputs a counterexample trace. In its original formulation, model checking uses finite state machines (or Kripke structures, transition systems, etc.) as program model and temporal logic as specification language. Given a state machine S and a property φ, a model checker determines if S is a model of φ, in symbols $S \models \varphi$. The major challenge in model checking is the high complexity of this question: Just note that n bits of memory result in 2^n system states. Similarly, 2 concurrent threads of n steps each give rise to $\binom{2n}{n} > 2^n$ schedules. Each schedule may result in a different value of a shared variable, and thus in a different state.

In the last 30 years, model checking has developed a large toolbox for practical verification. Many of the methods have been acquired from neighboring disciplines such as theorem proving, abstract interpretation, process algebra, and artificial intelligence. We list a few that are most relevant for the HYDRA proposal:

(a) Abstraction reduces S to a simpler system $h(S)$, s.t. $h(S) \models \varphi$ implies $S \models \varphi$. If $h(S)$ loses too much information, it may have "spurious" counterexamples which are infeasible in the original system. In other words, $h(S) \not\models \varphi$ although the property is true in the real model $S \models \varphi$. [43, 50]
(b) Predicate abstraction uses theorem provers and SAT/SMT solvers to define and compute h [70]; predicate abstraction was extended to liveness and concurrency [111, 20].
(c) Counterexample-guided abstraction refinement (CEGAR) [37, 88] iteratively computes predicate abstractions of increasing precision by analyzing spurious counterexamples using SAT/SMT solvers [16, 101].
(d) Partial order reduction reduces the number of schedules in the analysis of a concurrent system [69, 106, 126] to those which can potentially affect the specification.
(e) SAT and SMT solvers [21, 113] have become powerful enough to decide feasibility and verification conditions for loop-free code fragments with hundreds of lines of code.

As discussed in Part B1, p.4, model checking has produced two generations of tools: first generation tools such as SPIN and SMV with a clear semantics and a sound and complete model checking engine, and software model checkers which are more powerful but typically neither sound nor complete [17, 47, 15, 33]. The biggest challenge for verification of source code is shape analysis, i.e., abstraction and analysis methods for dynamic data structures. (The first generation makes use of (d) and (e) only, the 2nd generation of all methods (a)-(e).)

TLC [129], the model checker designed for the TLA+ language, is a typical representative of the first generation. Its input language is geared to distributed algorithms and concurrent systems. Internally, TLC uses explicit state exploration techniques, and uses sophisticated optimizations to deal with large state spaces (state hashing, disk storage, distributed exploration), but does not benefit from the breakthroughs (a)–(e). Consequently, TLC is heavily affected by *state explosion*, and can verify only systems with a few processes.

Our previous work on non-parameterized model checking (related to HYDRA)

(i) CEGAR. The PI participated in the invention of the CEGAR framework [37, 38] which was a cornerstone for the second generation of model checkers, cf. (c) above.

(ii) Termination and Bound Analysis. Recent years have seen a rapid advance in the automated analysis of termination [111, 46, 112, 48], liveness properties [45, 12] and bounds on resource consumption [71, 130, 115]. Our group has contributed scalable techniques for the bound analysis of real-world C# and C programs [71, 130, 115], the first CEGAR approach for constructing lexicographic ranking functions [48], and the first result on the decidability of liveness properties for parameterized timed networks [12].

Parameterized model checking

For distributed algorithms, the model checking techniques discussed above can verify only small instances with a **fixed** number of participating processes: If S_n denotes a system composed of n processes with shared memory, an off-the-shelf model checker can verify $S_N \models \varphi$ for some fixed number N, typically less than 10. (This

[3]As discussed on page 8, HYDRA does not target verification of real-time systems. We therefore omit a discussion of TIOA.

is what the TLA+ model checker TLC actually does.) To verify this distributed algorithm in full generality, we need **parameterized verification, i.e., verification for all values of** n. Let $S_{n,t}$ denote a system with n message passing processes of which at most $t < n/3$ can be faulty, and S_G denote the state machine for a distributed algorithm with a finite network topology graph G. Formally, the three verification questions can be stated as

$$\boxed{\forall n \geq 2 \,.\, S_n \models \varphi \qquad \text{and} \qquad \forall n \geq 2 \,.\, \forall t < \frac{t}{3} \,.\, S_{n,t} \models \varphi \qquad \text{and} \qquad \forall\, G.\, S_G \models \varphi}$$

respectively. Hence, parameterized model checking is, in a mathematically precise sense, strictly more difficult than ordinary model checking, and relies on advanced techniques from logic and automata theory. Although parameterized model checking was on the model checking research agenda since the early days ([41] even was published at PODC), it took significant time and effort to develop methodologies that fit the complexity of distributed algorithms. We finally avail of a rich toolset in parameterized verification; we mention those most relevant for HYDRA:

(f) The early **invariant-based techniques** [89, 128, 31, 95] required the user to provide an invariant candidate in the form of a labeled transition system. The technique then checked, whether the invariant candidate was strong enough to prove an inductive step on the number of processes. This check required to construct one form of an equivalence relation: a trace equivalence or a bisimulation. Later methods involved simulation and were applicable to more complex topologies expressible with network grammars [42, 83]. Invariants were used to check the following case studies: a hardware arbiter and Dijkstra's token ring [95]; mutual exclusion in token ring [128, 31].

(g) Cut-offs formalize the intuition that errors must occur already for a small number of processes: there exists a system of k processes (a cut-off) that satisfies a temporal logic formula if and only if every system of size $n \geq k$ satisfies the same formula. Cut-off results were established for very restricted computational models: token-passing systems [55, 39], systems with disjunctive and conjunctive guards [53], and FIFO systems [28]. Recent "dynamic" cut-off techniques apply to a broader class of systems, but do not guarantee to find a cut-off [81, 90, 6]. The following concurrent protocols were shown to have cut-offs, and thus shown to be verifiable with non-parameterized tools: MSI, MESI, MOESI, Illinois, Berkeley, N+1, Dragon, and Firefly cache coherence protocols [53, 54]; boolean abstractions of device drivers [81]; Burns's, Dijkstra's, Szymanski's mutual exclusion protocols [6].

(h) The technique of **invisible invariants** [13, 108, 99] exploits the small model property from logic to obtain a cut-off for a specific specification. Invisible invariants combined with ranking functions were also able to verify liveness properties [60]. Invisible invariants were used to check the following case studies: German's cache coherence [108, 99]; versions of simplified Bakery and Peterson's mutual exclusion [99].

(i) Counter abstraction techniques [74, 110, 18, 77] map a system S_n of n processes to a finite-state system which has counters to represent the number of processes in each state. To keep the counters finite, they are themselves abstracted. As in **(a)**, the mapping h loses information and introduces spurious behavior, which often violates liveness properties. Such unwanted behavior is excluded by adding justice and compassion conditions to the specification [110, 77]. Similarly to counter abstraction, environment abstraction overapproximates the behavior of a process environment [40, 121]. Counter and environment abstraction were used to check the following algorithms: Szymanski's and versions of simplified Bakery mutual exclusion algorithms [110, 40]; boolean abstractions of device drivers [18]; German's and FLASH cache coherence protocols [40, 121].

(j) Regular model checking (RMC) is an automata-based framework that represents a global state in a parameterized system as a word whose alphabet are local states [127]. The verification goal is to compute the transitive closure of the transition relation (which is given by an automaton) and check language intersection. Since the closure is not computable in general, RMC utilizes approximation techniques such as acceleration and widening [95, 1, 29, 109]. Since a global state is represented as a word, RMC can deal with simple topologies and asymmetries such as rings and lines. To verify liveness properties, RMC was generalized to RMC with Büchi automata [27, 24] and RMC with LTL(MSO) [8]. Further generalizations include RMC for context-free languages [67] and RMC over trees [7, 26]. Monotonic abstraction [5, 4, 2] extends regular model checking by employing the framework of well-structured transition systems [3]. Regular model checking and monotonic abstraction were used to check the following case studies: readers/writers, sleeping barbers, missionary and cannibals [4]; dynamic data structures [2]; token passing, tree arbiter, leader election [26]; versions of simplified Bakery, Burns', Dijkstra's, Szymanski's mutual exclusion [27].

(k) Compositional reasoning (CMP) reduces verification of a system with n identical processes to verification of a system with just 2 processes and thus allows the use of finite state model checkers [121, 100, 36]. CMP has been successfully used to formally verify cache coherence protocols used in some of Intel's multi-core processors [105].

(l) Proof-spaces [63] generalize inductive data flow graphs [62] for parameterized verification of shared memory programs. The proofs are constructed from predicate automata, an infinite-state generalization of alternating finite automata. For a survey of this automata-theoretic viewpoint, we refer to [61].

(m) Further related work on parameterized model checking includes broadcast systems [58, 56], atomic networks [59], lossy channel systems [3], ad hoc networks [51], well-structured transition systems [3, 64], networks of hybrid automata [79], and counter automata [94].

Model checking of fault-tolerant distributed algorithms

Completely automated model checking of fault-tolerant distributed algorithms is usually not parameterized [91, 78, 125, 119, 118]. Aside from our work, there were a few attempts on parameterized model checking of distributed algorithms: in [9] several synchronous broadcast algorithms were encoded by hand and verified in the MCMT model checker; in [66], the theoretical framework of regular model checking was extended to context-free languages — to capture conditions on the number of processes like $n > 3t$ — but, according to the authors, it has never been implemented and tried on fault-tolerant distributed algorithms.

Our previous work on parameterized verification

(i) Environment Abstraction [40, 44, 120] we discussed in "**(i) Counter abstraction**" above.

(ii) Network decomposition is a method to reduce a verification problem on a large network topology to multiple model checking calls on small networks. In [39], we have shown that token passing networks with arbitrary topology always have cut-offs, but the proof was not constructive and the cutoffs may not be computable. In recent work, [10] we computed cutoffs for some concrete topologies such as rings and cliques and extended the specification logic. In [11] we showed that cutoffs are computable for large classes of networks (technically, networks which are Monadic Second Order (MSO)-definable and have bounded clique-width). To achieve this result, we used a decomposition technique for MSO called the *Feferman-Vaught theorem* (see e.g. [98]). We also showed there that parameterized model checking for rendezvous systems and branching time specifications is undecidable, and therefore requires aggressive verification techniques such as abstraction.

In recent work we also studied logics for relational structures which can be used to reason over tree-like topologies. In [86] we showed the decidability of a description logic with trees; this logic allows us to reason about topologies such as stars or rings. In follow-up work [87], we showed a strong decidability result for the two-variable fragment of first order logic with counting and MSO-definable graphs of bounded tree-width.

(iii) Monograph on Decidability of Parameterized Verification. In the course of the above work, we realized that the theoretical literature on decidability of parameterized verification is quite scattered and sometimes incoherent. Thus, a paper intended as a survey gradually evolved into a monograph that will be be published later this year [22].

(iv) Parameterized Verification of Fault-Tolerant Distributed Algorithms (FTDAs).

In a series of papers [78, 77, 76, 84, 85] we developed the first scalable automated verification method for threshold-based fault-tolerant distributed algorithms; [4] to the best of our knowledge, this is the first example of a verification tool that can handle a class of advanced distributed algorithms with minimal manual intervention.

Our method applies to distributed algorithms like BOSCO discussed in Part B1 of this proposal. The characteristic feature of such algorithms are *threshold guards* which count messages and compare them against the total number n of processes and the number t of potentially faulty processes:

```
if received <ECHO> from at least n-t distinct processes then accept
```

In parameterized verification, these numbers have to be treated symbolically. Therefore, the most natural methods for this class such as counter abstraction are not directly applicable. To achieve verification, we proceeded in three steps:

(1) We modeled the algorithms in a parameterized extension of the SPIN input language PROMELA. This made it possible to verify fixed parameter instances in SPIN [78] and acquire assurance that our models are adequate: the model checker confirmed the theoretical results concerning correctness and resilience conditions.

(2) We introduced a *parametric interval abstraction* that enables the tool to reason about expressions such as $n - t$. Applying this abstraction to both the data domain and the counters in counter abstraction makes parameterized verification of non-trivial algorithms such as asynchronous reliable broadcast [116] possible [77]. To verify liveness properties in the spirit of counter abstraction [110], we developed a CEGAR loop which adds justice properties to eliminate unfair spurious counterexamples.

[4] Our paper at CAV 2015 [85] also was accepted in the Artifact Evaluation, which certifies reproducibility of the experiments.

(3) To increase scalability, we introduced static partial order reductions, obtained bounds on the diameter of the system, and employed SAT/SMT-based bounded model checking [84, 85]. This enabled us to verify safety of sophisticated fault-tolerant algorithms [85] such as asynchronous reliable broadcast (Distributed Computing 1987, JACM 1996), non-blocking atomic commit (Distributed Computing 2002, HASE 1997), condition-based consensus (DSN 2003), and one-step consensus (DSN 2006, DISC 2008).

Hence, this work serves as our main motivation for the HYDRA proposal. The success story of FTDAs is reflected in the HYDRA methodology of Figure 1.

Bottom line

The state of the art in parameterized model checking can address only very limited classes of distributed algorithms, with an overwhelming focus on safety properties of concurrent shared memory programs and cache coherence protocols.

This is reflected in the case studies in **(f)–(m)** such as Szymanski's mutual exclusion algorithm, simplified versions of the Bakery algorithm, MESI cache coherence, FLASH cache coherence, German's protocol, etc.

The lack of a common input language makes it hard to compare parameterized verification tools.

Our own recent work on parameterized verification of fault-tolerant distributed algorithms is a first step towards more complex distributed algorithms. We believe that the power of modern decision procedures, automata theory, partial order reductions and abstraction methods make similar success stories for other, broader classes of distributed algorithms possible.

(a.2) The gap between parameterized verification and distributed algorithms

The technical gap between the verification research and the reality of distributed algorithms [97, 14, 123, 32] directly affects the following standard features of distributed algorithms (introduced by example in Part B1):

1. **Message Passing.** Unless the processes in a distributed system have access to the same physical or virtual memory, the distributed algorithm consists of loosely coupled processes that exchange messages over (possibly buffered) communication channels. In the standard textbooks by Lynch [97] and Attiya and Welch [14], around 75% of the material is concerned with message passing algorithms. The algorithms in the literature use a variety of assumptions about delivery guarantees and timing.
2. **Fault Tolerance.** Fault tolerance is a core topic in distributed algorithms. Regardless whether the system is distributed due to non-functional reasons (e.g. replication of a safety critical application) or whether the system is distributed due to functional reasons (e.g., globally distributed servers, networks etc.), fault tolerance ensures that the system remains operational despite the faults. The correctness of the algorithm is typically tied to a specific fault model from a large class of possibilities. Realistic fault models classify the effect of the fault (e.g. message loss, process crash) and quantify the fraction of components that can be faulty using a *resilience condition*. The majority of the contemporary literature in distributed algorithms is concerned with fault tolerant algorithms.
3. **Communication Graph.** The communication channels between the processes of a distributed algorithm form the edges of a graph which represents a communication network. In some cases, the communication network is known to the processes, and the algorithms can exploit graph theoretic properties: This is the case for example for cliques, rings, trees and similar structures. In other cases, the graph is unknown to the processes, and the distributed algorithms have to compute a graph on top of, e.g. a spanning tree. The spanning tree then serves as a *virtual communication graph* for other algorithms.
4. **Partial Synchrony.** While distributed algorithms can be used for synchronization, the algorithms can usually not assume the system to be synchronous. On the other hand, a purely asynchronous setting has theoretical limits: The famous impossibility result by Fischer, Lynch and Paterson [65] shows that no distributed algorithm can achieve fault-tolerant consensus in an asynchronous system even for a single simple crash fault. The distributed algorithms research has created a rich variety of intermediate "partially synchronous" models with restrictions on maximum message delays and fairness assumptions for the scheduler. The correctness of many fault tolerant algorithms depends on specific forms of partial synchrony.
5. **Liveness.** Liveness is a central concern for distributed algorithms for two reasons: First, typical distributed algorithms spend most of their time waiting for a message to arrive; thus, they have wait cycles whose termination depends on external events. Second, the presence of faults may affect reliable communication, e.g. processes may crash or fail to send messages. Fault-tolerant algorithms have mechanisms which ensure the termination of wait cycles despite faults and crashes. Consequently, the liveness of a fault-tolerant algorithm is both mission critical and difficult to prove.

6. **Process IDs.** In realistic distributed computing, all processes have process IDs. This is also reflected in distributed algorithms: Even simple tasks such as leader election are impossible without the use of process IDs. (This is the first and the simplest impossibility result in [14].) IDs are not just used to identify specific processes via their name, but to break symmetries between otherwise isomorphic processes. Consequently, the majority of distributed algorithms assumes distinct process IDs, which are sometimes linearly ordered.
7. **Data Structures.** Distributed algorithms use a large variety of data structures including integers, buffers, lists, queues, trees, tuples and sets along with the standard operations on these data structures. Since the number of processes in a distributed algorithm is arbitrarily large (a parameter in terms of model checking), the data structures are typically also unbounded: they have to store process IDs, broadcast messages, count messages etc. Often, the pseudocode description of an algorithm keeps some of the data structures implicit, e.g. the collection of hitherto received messages.
8. **Signatures.** Certain distributed algorithms make use of cryptographic primitives such as signatures to certify the origin of a message in the presence of possible faults. The increased assurance may result in a better resilience condition, i.e., a larger number of possible faults. The underlying cryptography is typically assumed to be perfect in the sense of the Dolev-Yao model [52].
9. **Real Time and Hybrid Systems.** Fault-tolerant distributed algorithms are often used in embedded systems that have to interact with a physical environment. This necessitates to include an analogous model for the environment, most importantly, a physical notion of time. The assumptions for distributed algorithms range from simple time bounds (e.g. message delays) all the way to complex notions from control theory.
10. **Probabilistic Behavior.** Distributed systems exhibit lots of nondeterminism due to the scheduler and the faults. The processes themselves are usually deterministic, i.e., they can make no non-deterministic choices themselves. This weaknesses of the processes leads to the impossibility results discussed above. One way out of the impossibility results is to give the processes access to a random number generator. Thus, one can obtain algorithms which are able for instance to break, with high probability, the symmetry between processes without IDs.

It is impossible to develop verification methods for 40 years of research in distributed algorithms in a single ERC grant. To achieve maximum impact, we will focus and prioritize our research directions and our choice of methods in accordance with the challenges discussed above and the expertise needed to address them.

(a.3) Objectives

The vision of HYDRA is to develop, for the first time, an automated verification framework that is able to verify a significant class of realistic distributed algorithms.

This gives rise to four objectives which will structure and guide our work:

The ***high-level objective*** is to improve the quality of critical distributed systems through rigorous algorithm design that is tightly coupled with automated verification methods. Our vision are formally verified fault-tolerant distributed algorithms which are immune against both faults and logical design errors. Domain specific modeling languages will bridge the gap between algorithm design and tool-supported analysis.

The ***theoretical objective***, and main challenge, is to permeate and exploit the mathematical structure inherent in distributed algorithms to make them amenable to verification. Informed by the manual proof and construction principles from distributed algorithm theory, we will develop specific automata models, partial order reductions, abstractions, decomposition and quantitative methods for use in verification algorithms.

The ***practical objective*** is to develop a model-checking based toolchain that is able to analyze a large class of distributed algorithms. To harness the full potential of state-of-the-art model checking, we will combine powerful existing tools (for instance, SAT and SMT solving) with the rich theoretical knowledge in parameterized verification. In the spirit of almost automated verification, the tools should be fully automated wherever possible, and allow human guidance where necessary.

The ***social objective*** is to foster systematic collaboration and exchange between the computer-aided verification community and the distributed algorithms community. Through the establishment of an interdisciplinary team, competitions, an algorithms repository, and strategic collaborations with researchers from both communities we will stimulate a culture of collaboration.

While we cannot solve all technical problems in the 5 years of the ERC grant, we will achieve a critical mass of results that will establish parameterized model checking of distributed algorithms as a research discipline in its own right. Given the number and complex relationship of these challenges, it appears very difficult to make tangible progress without the large interdisciplinary group made possible by an ERC grant (15 postdoc years + 15 PhD student years).

Despite the importance of message-passing algorithms explained above, most of the research activity in parameterized verification has been concentrating on *shared memory concurrent programs without faults*, cf. (a.2) above. In contrast, our own recent threads of research on (i) fault-tolerant distributed algorithms with clique topology and (ii) network decomposition have gone significantly beyond this model.

For the HYDRA project, we therefore take the strategic decision to focus on algorithms which combine fault tolerance and communication graphs with the additional features (1)–(10) discussed above:

	No faults	**Fault tolerance**
Cliques	concurrent programs	*HYDRA area of expertise*
Complex graph classes	*HYDRA area of expertise*	*HYDRA challenges*

Our focus on message-passing and fault tolerance leaves out concurrent programs and their "hot" research areas such as weak memory models and eventual consistency, but advances the state of the art towards comprehensive coverage of distributed algorithms. We will of course benefit from the existing body of research about concurrent programs, and quite possibly also contribute to advanced questions such as liveness, but will not focus our efforts in this direction. *Thus, we will use the ERC grant to explore entirely new research areas rather than compete on verification of concurrent programs with other groups.*

(b) Methodology

In this section, we systematically address the high-level objective, the practical objective, and the theoretical objective in accordance with Figure 1. We return to the social objective at the end of the section.

Mathematical Artifacts. The central column of Figure 1 describes the technical workflow which leads from a syntactic model – i.e., a program that is natural to understand for a distributed computing specialist – to a precise semantic model, i.e., a mathematically precise model of the distributed algorithm. This model is understood by verification tools, but too complex to analyze directly. We use methods such as abstraction, partial order reduction, and decomposition to reduce the semantic model to a simpler model that is either decidable or has good algorithmic properties in practice.

Tools and Modeling Frameworks. For each stage, the left column illustrates the tools that we will develop, and the right column the languages that are used by the tools. We do not expect that a single language will be accepted by all application areas. We start with established languages such as IOA and PlusCal, will extend them to capture additional features such as fault tolerance, and complement them by domain specific languages. As discussed below, we will coordinate our language definition efforts with colleagues working on related problems. We will develop configurable *translator tools* that compile the syntactic models into semantic models, and transformations between the models. The semantic model is formulated in accordance with the paradigms of verification, i.e., by suitable automata models and logical formalisms. The translation from the syntactic model to the semantic model has to capture the intended semantics of an algorithm that was previously described in pseudocode. Thus, it is important to obtain confidence that the syntactic model is an *adequate* representation of the distributed algorithm. A *non-parameterized model checker* then helps to find obvious errors in the model, and can check conformance of the distributed algorithm with theoretical predictions for a fixed number of processes.[5] The *simulator* uses the model checker to facilitate interactive exploration of the model similar to a debugger. Due to the enormous non-determinism in the distributed algorithms, the simulation steps are essentially model checking questions. The final step – and the dominant part of the research agenda in HYDRA – is the development of parameterized model checking methods and tools.

Almost Automated Verification. The lower half of Figure 1 suggests that parameterized model checking uses complex *internal* tool chains, i.e., tools such as theorem provers, SAT and SMT solvers, decision procedures, and model checkers that are hidden from a user who is just interested in distributed algorithms. In reality, this

[5]This is what TLC is currently used for; as discussed above, TLC has many technical restrictions to be overcome.

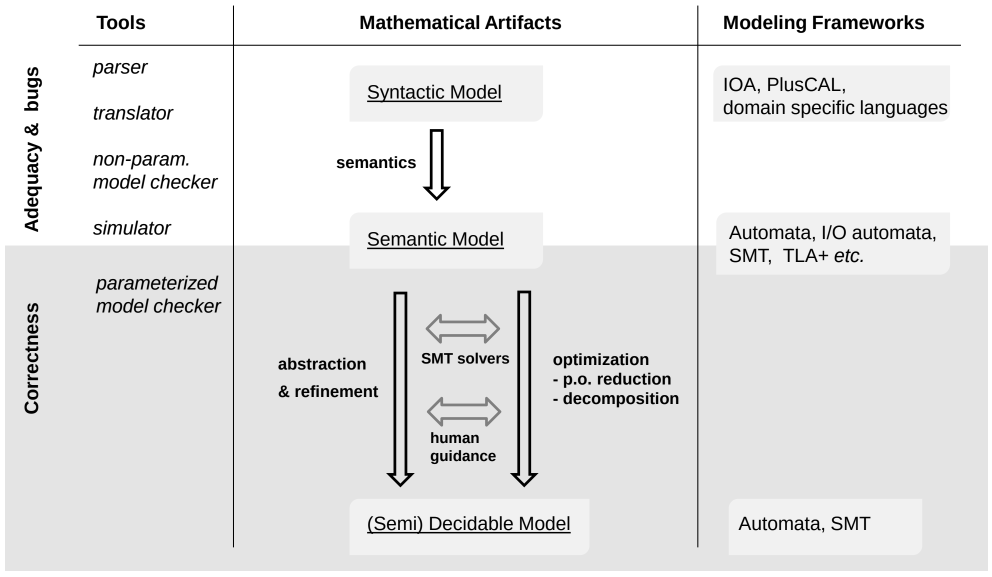

Figure 1: Overview of the methodology and tool chain

is an idealization: abstraction functions, invariant candidates, annotations and tool parameterizations are often a crucial input to make verification work in practice. Set up the right way, these auxiliary informations *do not interfere with the soundness of the verification* – for instance, a "bad" abstraction function will not lead to false positives, but only to spurious counterexamples. In fact, our own recent work on FTDAs requires some manual intervention to define an abstraction function. This is not uncommon in situations where the correctness of an algorithm or design is more important than the simplicity of the tool chain. For instance, the verification of cache coherence protocols also relies on auxiliary lemmas from the user [100]. Our goal for the verification of distributed algorithms is to find a good balance between automation where possible and human guidance when necessary: We refer to this approach as Almost Automated Verification. Considering the rich mathematical structure behind distributed algorithms and the relatively small size of the algorithms themselves, we believe that this is the most reasonable and pragmatic approach.

We expect that Almost Automated Verification naturally leads to fully automated verification in the long term: Setting up the verification process in a way where auxiliary information never interferes with the soundness of the procedure, but only affects the efficiency, we can apply aggressive methods including heuristics to *compute* the auxiliary information. A paradigmatic demonstration of this principle is the best paper award winning paper at FMCAD 2008 by Talupur and Tuttle from Intel who use informal diagrams from design documents to support a formal verification process [121].

Verification Themes

We will address the gap between the state of the art in parameterized verification and the features of distributed algorithms in five Verification Themes that we shall discuss now. Since the Verification Themes have complex dependencies, and the ERC grant will support a large team, we will work on all five Themes concurrently, typically considering challenges from two Themes at a time. We will thus continue our successful strategy of stepwise improvement: Starting from simple case studies and existing tools, we extend the realm of modeling and algorithmic verification with additional features of distributed algorithms. Each step is typically motivated by a benchmark example from practice or from the research literature.

As Figure 1 indicates, Modeling and Tool Construction are central concerns that unify the Verification Themes. The coherence of models and interoperability and maintenance of tools is a major concern for all verification projects, and will be coordinated both by the PI and a postdoc researcher; cf. also the discussion below on the high-level and social objectives.

Considering the competence and expertise of our group in logic and verification of *discrete* systems, we will *not* directly focus on features 9 and 10 – real time / hybrid systems and probabilistic systems – in the Verification Themes of the HYDRA project. Thanks to our ongoing collaboration in the RiSE national research network, we expect synergies and joint work in hybrid and real time systems, but we do not promise to take a lead in these tasks. Synthesis is not part of HYDRA for a related reason: Both Roderick Bloem and the PI have RiSE tasks on synthesis of distributed algorithms which complement the research agenda of HYDRA in a natural way. We will discuss these issues after the description of the Verification Themes.

Verification Theme 1: Communication in the presence of faults

As discussed in Section (a.1), our previous work [77, 84, 85, 76] resulted in the first parameterized verification of distributed algorithms with a realistic fault model. The distributed algorithms we considered were simple in the sense that the messages did not explicitly carry information – we used different message types from a finite collection to simulate messages with a few bits of information. It is evident that this approach does not scale to messages that can carry complex data of possibly unbounded size. (In a parameterized setting, already a single process ID will be unbounded in size.) Moreover, we assumed the communication channel to be reliable, and only the processes to be faulty. In HYDRA, we will consider both fault models for the communication channels and complex messages.

The semantics of faulty communication channels is both a modeling and a verification challenge. We need to account for different types of channels and faults such as FIFO buffers, fair lossy links, eventually reliable links etc. The easiest way to describe the communication semantics to a model checker is to add a message manager with the appropriate functionality. Essentially, this means to build a harness for the distributed algorithm. The challenge then is the complexity of the message handler: we have to verify a distributed algorithm *together* with a message handler that may be more complex than the algorithm itself. There is, however, a separation of concerns: since the message handler is independent of the distributed algorithm, we can study different kinds of message handlers case by case and develop tailored methods, in particular partial order reductions. Our work on FTDAs can also be understood from this perspective.

To capture complex messages, we will model them as symbolic terms in a logical language. Often, we are not interested in the specific value of a message, but in the relationship of the value with respect to other variables, e.g. to two local maxima. Such observations make it possible, similar as in environment abstraction, to apply variations of predicate abstraction. We may also introduce global shadow variables and relate the symbolic values to global invariants involving these variables. An important challenge is set comprehension: How can we symbolically represent sets of symbolic messages, measure their cardinality, and select or compute information from them? This question is naturally addressed in connection with Theme 5 below. As the symbolic message model is similar to the Dolev-Yao model in computer security, this will also enable us to model signed messages in a way that is compatible with existing research in computer security.

Verification Theme 2: Partial synchrony

From a model checking perspective, partial synchrony can be viewed as a form of fairness assumption which gives quantitative guarantees for progress (non-starvation) of a process. Thus, partial synchrony is a *sine-qua-non* condition for the liveness properties of many algorithms, cf. Theme 3.

Note that partial synchrony *of process schedules* can also be seen as a restriction on the set of asynchronous schedules, and is therefore technically related to partial order reduction: potentially, the same techniques and program instrumentations which restrict the schedule for partial order reduction can be used to remove schedules which violate partial synchrony. We will therefore revisit the partial order reductions that we developed for FTDAs from the perspective of partial synchrony. Since partially synchronous algorithms exhibit less non-determinism from interleavings than asynchronous ones, there is a potential that they are computationally easier to verify, but the partial order reduction itself is technically much more involved. The ultimate goal is a partial order reduction framework which encompasses different notions of partial synchrony as special cases.

Partial synchrony *of message delivery* refers to quantitative delivery guarantees for messages. It strongly depends on the interactions of the process scheduler and the message handler discussed in Theme 1. To address this question, we first need to solve the more basic challenges for Themes 1 and 2 discussed above.

Verification Theme 3: Liveness and quantitative bounds

As liveness only guarantees the progress of a system in very abstract terms, a quantitative bound is important to understand the performance of the algorithm. Unfortunately, there are very few results on verification of liveness properties for parameterized systems, and even fewer on time bounds. A notable example is the work [110] that introduces fairness (justice and compassion) in the course of abstraction, and thus allows one to verify liveness. Our recent work [77] uses ideas from [110] and is the first to prove liveness properties of purely asynchronous FTDAs. Since our own research group contributed to the recent advance in automated termination and bound analysis (see section on state-of-the-art), we are in a good position to generalize this work to parameterized verification. To this end, we will proceed in two steps:

First, we will study liveness of parameterized systems in a simple synchronous framework where each process is guaranteed to make a step at each clock cycle, i.e., there is steady local progress. To show termination, we need a global ranking function that is parameterized in the number of processes. The counters of a counter abstraction lend themselves naturally as components for the synthesis of ranking functions. We can then use the ranking functions to extract time bounds following our previous work [71, 130, 48, 115].

Second, we will generalize our work from synchronous systems to partially synchronous ones which may or may not have message delays or faults. Since partially synchronous systems trade off the guarantee of local progress at each step against a progress guarantee over a longer time window, the technical challenge is to align the ranking function to this longer time window, for instance to have a ranking function which includes history information.

Note that many of the challenges in current work on source code verification of liveness and bounds is motivated by full automation for large programs with thousands of lines of code. Thus, the underlying theory is quite well understood, but the implementation for real life code is challenging. In the context of almost automated verification pursued in HYDRA, we can rely on the user to provide auxiliary information such as components of the ranking function. We are thus optimistic that liveness and bound analysis are well in reach with current methods.

Verification Theme 4: Communication graphs and process IDs

On a p2p network without process IDs (i.e., a clique), all communication has to be either broadcast or nondeterministic: there is no possibility to send a message to a specific recipient. This is an advantage for verification, because the absence of IDs creates symmetries which we can exploit in the verification algorithms. The presence of process IDs makes it possible to communicate with specific processes. Thus, even in a clique communication graph, the distributed algorithm can *de facto* establish a virtual communication graph on top of the clique, e.g., process i is connected to processes $i-1$ and $i+1$ resulting in a ring topology, or everybody is connected to the process with minimum ID resulting in a star etc. Thus, the presence of IDs is technically very similar to the presence of a communication graph. For the classes of communication graphs obtained in this way, it is important how IDs are used and manipulated in the algorithm: whether they are just compared for equality, compared as numbers, or used in arithmetical computations. Depending on the use of IDs, it is thus natural to model the domain of process IDs as pure sets, linearly ordered sets, or in Presburger arithmetic, and study the virtual communication graphs they can define. Consequently, we will study distributed algorithms with process IDs in the context of distributed algorithms with communication graphs.

Our previous work has used strong results from model theory (such as the Feferman-Vaught theorem) to decompose large communication graphs into smaller communication graphs that can be verified individually [11], cf. p. 4 (ii). This work however is limited to simple finite state processes and communication by token passing. The CMP method by McMillan, Tuttle, Talupur [100, 121] in contrast achieves practical compositional reasoning by focusing on few selected processes, and abstracting the others as environment; note that CMP is able to handle process IDs, but usually requires manual auxiliary lemmas. To achieve verification on complex communication graphs, we need to combine the CMP method with our network decomposition approach. To this end, we will pursue two strategies that will converge in the long term: (i) We will gradually extend CMP, first to simple classes of communication graphs, e.g. trees and rings, and then to a more general setting. (ii) We will further develop our decomposition techniques to more realistic communication models and larger classes of communication graphs for which we either have decidability or a good algorithmic understanding. Murali Talupur has expressed strong interest to collaborate on this topic.

Verification Theme 5: Complex Data structures

In ordinary program analysis, shape analysis is a big challenge, because the data structures are unknown, they form complex graphs, they use pointer arithmetic, and they do destructive updates which compromise the integrity of a data structure. For distributed algorithms, the situation is different: the pseudocode uses natural high-level operations on generic data structures such as sets, lists, buffers, trees etc. whose implementation details are not discussed, and not important for the correctness of the algorithm. Our hypothesis is that this aspect of data structures is simpler in distributed algorithms than in source code. In fact, many data structures that occur in distributed algorithms can be modeled as multisets of tuples: Lists, buffers, queues are typically just used to store messages and events in the order of their arrival, and can be viewed as multisets whose elements have time stamps.

Distributed algorithms often compare cardinalities of sets, e.g. to conclude that the majority of processes has sent a certain message. This is often central to the fault tolerance of the algorithm, and connected to the resilience condition, i.e., the number of possible faults. They also choose elements with certain properties from sets to break symmetries, for instance they choose the maximum process ID, or the minimum time stamp. The main challenge for verification is the unboundedness of the data structures: a buffer which stores messages from

N processes can have up to N elements. Thus, for parametric N, the buffer size is symbolic. Consequently, cardinality comparisons and selection of elements also have to be symbolic. The challenge thus is to have a symbolic representation of multisets which is compatible with parameterized verification. One possibility is to introduce shadow variables which explicitly keep track of the cardinality, the minimum, or maximum instead of manipulating the set. In our work on FTDAs discussed on p.4, we introduced message counters for cardinalities, and introduced a symbolic interval abstraction with a finite state space for the message counter and for counting the cardinalities of processes that enter a certain state. Together, this made it possible to apply counter abstraction, and thus to verify the algorithm; the challenge is to identify such situations automatically.

Finally, to achieve automated abstraction for broader classes of algorithms, we need a good logical understanding of multisets. We believe that the theory of "data words" that has been studied extensively in database theory and finite model theory [30, 25, 103] will provide technical tools to handle multisets in a more general setting, and also to reason about more complex message content. Our previous work in finite model theory will help us to integrate these methods into verification.

Collaboration with the RiSE network: real time, probabilities, and synthesis

The National Research Network RiSE (Rigorous Systems Engineering) is a large strategic grant by the Austrian science foundation. Coordinated by Roderick Bloem (speaker) and Helmut Veith (vice-speaker), RiSE is concerned with new verification paradigms with an emphasis on synthesis. RiSE has a systematic collaboration with the doctoral college PUMA in Munich, the doctoral college LogiCS in Austria, and the NSF expedition grant ExCAPE in the US. In 2014, RiSE was extended for the second (and final) funding period until 2018. Note that our verification breakthrough for FTDAs was achieved in the first funding phase of RiSE. The topics in the second funding phase of RiSE are synergetic but *disjoint* with HYDRA. RiSE is complementing the research agenda of HYDRA with regard to synthesis, real time and probabilistic systems:

(i) Synthesis. Roderick Bloem, Tom Henzinger and Helmut Veith have tasks concerned with the synthesis of distributed and concurrent systems. New HYDRA results on verification will have strong impact on the synthesis tasks.
(ii) Probabilistic and Timed Systems. Ezio Bartocci is leading a RiSE task on parameterized verification of probabilistic timed systems, Radu Grosu a task on probabilistic analysis of distributed systems, and Ana Sokolova (Salzburg) a task on trace semantics for probabilistic systems.

HYDRA and the RiSE network have strong synergies in synthesis, probabilistic, hybrid and timed systems, but no double funding.

Revisiting the high-level and social objectives: repositories, competitions, workshops, collaborations

The HYDRA project combines a technical challenge with a community building effort. To foster the use of specification languages for verification of distributed algorithms, we will provide an *online repository* for distributed algorithms. Throughout the project, we will require all project participants to specify distributed algorithms in existing and novel specification languages. (This also has the important effect that students and postdocs with a background in formal methods will be immersed in distributed algorithms.) Thus, we will obtain an empirical basis for benchmarking and tool development, and increase the interest of other research groups from both involved communities.

Towards this goal, we will also initiate an annual competition for parameterized verification that will cover the range from concurrent programs to advanced distributed algorithms in multiple categories. We will associate the competition with FRIDA, an annual workshop on formal methods for distributed algorithms co-initiated by our group which is colocated with major conferences in model checking and distributed algorithms. To be widely accepted, the competition and the input languages should be a *community effort*. Armin Biere has offered his experience from the SAT and HW competitions for the organization of the new competition. By 2020, we expect to have collected hundreds of distributed algorithm models in the open repository.

We also plan to organize three Dagstuhl-style workshops in Vienna (or Dagstuhl) in Years 2016, 2018, 2020. In Vienna, the workshops will be cofinanced and supported by the Vienna Center of Logic and Algorithms. To sustain this activity beyond the HYDRA project, we will apply for a European COST network between distributed algorithms, verification, and application fields of distributed computing.

The PI has been lobbying for a colocation of CAV and PODC, and will continue his efforts.

Further collaborations

Industrial collaborations include Amazon (Byron Cook), TTTech (Wilfried Steiner). We will continue collaboration with Azadeh Farzan (Toronto), Murali Talupur (Portland), Tomas Vojnar (Brno), Sagar Chaki (CMU).

Software Artifacts

As HYDRA will produce multiple software artifacts (parsers, model checkers, simulators, translator tools), we will put special effort into making the tools available for other research groups. Thus, we will release the source code under an open source license, and will provide systematic documentation. In particular, we will ensure that the project will end with a final software release from which other researchers (and, of course, our own follow-up projects) will start. We will follow the example of the SPIN tool that is of high quality *and* well documented. Thus, in conjunction with the benchmark repository, our research will be reproducible. As HYDRA has the ambition to change the way we analyse distributed algorithms, a badly supported tool chain would undermine our efforts.

Project Organization

Like many projects in computer-aided verification, HYDRA has a theoretical side and an engineering side which roughly correspond to the practical and theoretical objectives discussed above. For the engineering aspects of tool building, modeling, and community building we plan the following milestones:

Month	*Milestone*
2	1st yearly retreat: tutorials for new HYDRA group members (repeated yearly)
6	Reproduction of experiments with existing tool chains for IOA, TLA+ and verification tools
8	1st HYDRA Workshop on *modeling languages, real life benchmarks, competition setup*
12	Collection of domain specific modeling languages
16	Parser and translator tools for all modeling languages
	Repository goes online with 30+ algorithms, 1st competition announced (competition repeated yearly)
23	1st release of non-parameterized model checker (“Cerberus”) and simulator (“Fluffy”)
35	1st release of the HYDRA parameterized model checker
36	2nd HYDRA workshop: *revisiting the challenge. what is needed for verification of distributed algorithms?*
41	2nd release of Cerberus and Fluffy
59	Final release of HYDRA, Cerberus and Fluffy
60	3rd HYDRA workshop, kick-off for COST network

For simplicity, the table lists repeated events like the yearly competition and retreats only once. To ensure coherence of models and tools in the HYDRA team, all project members contribute to these tasks. Modeling and tool development are coordinated by one postdoc each.

Concerning the theoretical objectives, the 5 students (3 funded by HYDRA plus 2 funded by the university and the doctoral college LogiCS, cf. Team on p. 13) roughly correspond to the 5 verification themes. The postdocs are chosen to provide expertise in distributed algorithms, model checking, and logic, and are involved in multiple Verification Themes. We shall work on all 5 Verification Themes in parallel *for 60 months* – moving from simple models to increasingly complex ones. The advanced verification features of the tools will go hand in hand with theoretical progress. The intertwined milestones for the fixed parameter tools (Cerberus & Fluffy[6]) and parameterized tools (HYDRA) are important: They guarantee that the theoretical research is tied to realistic models that were developed and tested using the simpler non-parameterized simulation and verification tools. As we need to build a lot of infrastructure for the 1st release of Cerberus & Fluffy, we expect the ratio between tool development and foundational research to be 1:1 in the first 2 years of HYDRA, and 1:3 in the last 3 years. Thus, we will escape the dilemma of parameterized verification discussed in Part B1: the negative reinforcement between lack of benchmark models and lack of powerful model checkers.

Why is HYDRA a high-risk / high-gain effort?

If successful, the HYDRA project will revolutionize the use of computer-aided verification in distributed algorithms and distributed computing: It will be attractive for distributed algorithms researchers to use the HYDRA tools for specification, simulation, verification and quantitative analysis of their algorithms. The availability of realistic models and benchmarks will be attractive for researchers in model checking and logic. Together, they will help to improve the quality and reliability of distributed algorithms in multiple application areas including cloud computing, many-core computation, ubiquitous computing, hardware design, the internet of things, and autonomous embedded systems. Thus, the proposed collaboration between Parameterized Model Checking and Distributed Algorithms is a win-win situation for both disciplines with significant impact beyond the two disciplines. In the long term, HYDRA will facilitate synthesis of distributed systems from verified models.

The HYDRA project is high-risk because it is technically difficult, and because the interaction between the two communities has been held back by insufficient model checking technology in the past. We believe that, based on our track record and recent work, we are well-positioned to undertake the challenge, and we are confident that the project will result in far reaching results that are exciting for both research communities.

[6]While the Hydra has an unbounded number of heads, the Cerberus and Hagrid’s Fluffy have three heads only.

(c) Resources

Cost Category			**Total in Euro**
Direct Costs	**Personnel**	PI	---
		Senior Staff	---
		Postdocs (3 persons, 5 years)	1039893
		Students (3 persons, 5 years)	687411
		Other (10h per week administrative assistant)	47242
	i. Total Direct costs for Personnel (in Euro)		1774546
	Travel		192500
	Equipment		---
	Other goods and services	Consumables	---
		Publications (including Open Access fees), etc.	12000
		Other (Audit Certificate)	20000
	ii. Total Other Direct Costs (in Euro)		224500
A – Total Direct Costs (i + ii) (in Euro)			1999046
B – Indirect Costs (overheads) 25% of Direct Costs (in Euro)			499762
C1 – Subcontracting Costs (no overheads) (in Euro)			---
C2 – Other Direct Costs with no overheads (in Euro)			---
Total Estimated Eligible Costs (A + B + C) (in Euro)			2498808
Total Requested Grant (in Euro)			2498808

For the above cost table, please indicate the % of working time the PI dedicates to the project over the period of the grant:	**40 %**

We request 3 postdoc positions, 3 PhD student positions, and a part-time administrative position. The salary rates follow the Austrian collective bargaining agreement. In accordance with the Austrian research agency FWF, PhD students are hired for 30 hours per week. For conference travel, we calculated 4500 EUR per year for the students, the postdocs, and the PI. In addition, we calculated 7000 EUR per year for travel costs of 3-4 academic visitors to Vienna each year. These numbers reflect typical travel costs in previous projects.

The following salaries will be financed by the university and by the Doctoral college LogiCs (see Part B1): the salary of the PI (including two sabbaticals in 2016 and 2020), two additional PhD students, administrative support, IT support, PR support.

Local Team. We will continue our successful collaboration at TU Wien with Ulrich Schmid and Josef Widder (distributed algorithms), Igor Konnov (model checking), and Tomer Kotek (logic) who have been core contributors to our breakthrough papers on parameterized verification of FTDAs. The postdoc salaries of the latter two will be financed by HYDRA. Josef Widder has funding for his own position and one student from his own stand-alone project on FTDAs (PRAVDA) for four years. Thus, we have an opportunity to hire an additional postdoc with a background in distributed algorithms at project start.

The FORSYTE group is headed by the PI, and has two tenure track assistant professors: Florian Zuleger, who will contribute to Verification Theme 3, and Georg Weissenbacher, who has a strong background in model checking and decision procedures. Moreover, Laura Kovacs has accepted a full professor position in the FORSYTE group, and will start an ERC Starting Grant on interpolation and theorem proving in 2016.

The FORSYTE group is very international, and we have more than 50% female graduate students. Combining the strong local tradition in logic with best practice examples in graduate education from world leading institutions, we have established a collaborative culture that fosters enthusiasm for research and offers a good combination of scientific autonomy and support for individual needs. We are pleased to see that our efforts since 2010 have increasingly attracted excellent international students and researchers to Vienna. HYDRA will enable us to keep the momentum.

Literature (abridged due to page limit)

[1] P. A. Abdulla, A. Bouajjani, B. Jonsson, and M. Nilsson. Handling global conditions in parametrized system verification. In *CAV*, pages 134–145. Springer, 1999.

[2] P. A. Abdulla, J. Cederberg, and T. Vojnar. Monotonic abstraction for programs with multiply-linked structures. *International Journal of Foundations of Computer Science*, 24(02):187–210, 2013.

[3] P. A. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *LICS*, pages 313 –321, 1996.

[4] P. A. Abdulla, Y.-F. Chen, G. Delzanno, F. Haziza, C.-D. Hong, and A. Rezine. Constrained monotonic abstraction: a CEGAR for parameterized verification. In *CONCUR*, pages 86–101, 2010.

[5] P. A. Abdulla, G. Delzanno, and A. Rezine. Monotonic abstraction in parameterized verification. *Electronic Notes in Theoretical Computer Science*, 223:3–14, 2008.

[6] P. A. Abdulla, F. Haziza, and L. Holík. All for the price of few. In *VMCAI*, pages 476–495, 2013.

[7] P. A. Abdulla, B. Jonsson, P. Mahata, and J. d'Orso. Regular tree model checking. In *CAV*, pages 555–568, 2002.

[8] P. A. Abdulla, B. Jonsson, M. Nilsson, J. d'Orso, and M. Saksena. Regular model checking for LTL(MSO). *STTT*, 14(2):223–241, 2012.

[9] F. Alberti, S. Ghilardi, E. Pagani, S. Ranise, and G. P. Rossi. Universal guards, relativization of quantifiers, and failure models in model checking modulo theories. *JSAT*, 8(1/2):29–61, 2012.

[10] B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI*, volume 8318 of *LNCS*, pages 262–281, Jan. 2014.

[11] B. Aminof, T. Kotek, S. Rubin, F. Spegni, and H. Veith. Parameterized model checking of rendezvous systems. In *CONCUR*, volume 8704 of *LNCS*, pages 109–124. 2014.

[12] B. Aminof, S. Rubin, F. Zuleger, and F. Spegni. Liveness of parameterized timed networks. In *ICALP*, page to appear, 2015.

[13] T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. D. Zuck. Parameterized verification with automatically computed inductive assertions. In *CAV*, pages 221–234, 2001.

[14] H. Attiya and J. Welch. *Distributed Computing*. Wiley, 2nd edition, 2004.

[15] G. Balakrishnan, M. K. Ganai, A. Gupta, F. Ivancic, V. Kahlon, W. Li, N. Maeda, N. Papakonstantinou, S. Sankaranarayanan, N. Sinha, and C. Wang. Scalable and precise program analysis at NEC. 2010.

[16] T. Ball, B. Cook, V. Levin, and S. K. Rajamani. SLAM and static driver verifier: Technology transfer of formal methods inside microsoft. In *IFM*, volume 2999, pages 1–20. Springer, 2004.

[17] T. Ball, V. Levin, and S. K. Rajamani. A decade of software model checking with SLAM. *Commun. ACM*, 54(7), 2011.

[18] G. Basler, M. Mazzucchi, T. Wahl, and D. Kroening. Symbolic counter abstraction for concurrent software. In *CAV*, pages 64–78, 2009.

[19] B. Batson and L. Lamport. High-level specifications: Lessons from industry. In *Formal methods for components and objects*, pages 242–261. Springer, 2003.

[20] J. Berdine, T. Lev-Ami, R. Manevich, G. Ramalingam, and M. Sagiv. Thread quantification for concurrent shape analysis. In *CAV*, pages 399–413, 2008.

[21] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[22] R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. *Decidability of Parameterized Verification*. Synthesis Lect. on Dist. Comp. Theory. Morgan & Claypool Publishers.

[23] A. Bogdanov, S. J. Garland, and N. A. Lynch. Mechanical translation of I/O automaton specifications into first-order logic. In *FORTE*, pages 364–368, 2002.

[24] B. Boigelot, A. Legay, and P. Wolper. Omega-regular model checking. In *TACAS*, pages 561–575, 2004.

[25] M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *LICS*, pages 7–16. IEEE, 2006.

[26] A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract regular tree model checking. *Electr. Notes Theor. Comput. Sci.*, 149(1):37–48, 2006.

[27] A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *CAV*, pages 372–386, 2004.

[28] A. Bouajjani, P. Habermehl, and T. Vojnar. Verification of parametric concurrent systems with prioritised FIFO resource management. *Formal Methods in System Design*, 32(2):129–172, 2008.

[29] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *CAV*, pages 403–418, 2000.

[30] P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data languages and timed languages. *Inf. Comput.*, 182(2):137–162, 2003.

[31] M. C. Browne, E. M. Clarke, and O. Grumberg. Reasoning about networks with many identical finite state processes. *Inf. Comput.*, 81:13–31, 1989.

[32] C. Cachin, R. Guerraoui, and L. Rodrigues. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[33] S. Chaki, E. M. Clarke, A. Groce, S. Jha, and H. Veith. Modular verification of software components in c. *Software Engineering, IEEE Transactions on*, 30(6):388–402, 2004.

[34] K. M. Chandy and J. Misra. Parallel programming design. *Addison-Wesley*, 1988.

[35] B. Charron-Bost, H. Debrat, and S. Merz. Formal verification of consensus algorithms tolerating malicious faults. In *SSS*, volume 6976 of *LNCS*, pages 120–134, 2011.

[36] C.-T. Chou, P. Mannava, and S. Park. A simple method for parameterized verification of cache coherence protocols. In *FMCAD*, volume 3312 of *LNCS*, pages 382–398. Springer Verlag, 2004.

[37] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV*, volume 1855 of *LNCS*, pages 154–169. 2000.

[38] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, Sept. 2003.

[39] E. Clarke, M. Talupur, T. Touili, and H. Veith. Verification by network decomposition. In *CONCUR 2004*, volume 3170, pages 276–291, 2004.

[40] E. Clarke, M. Talupur, and H. Veith. Proving Ptolemy right: The environment abstraction framework for model checking concurrent systems. In *TACAS*, pages 33–47. Springer, 2008.

[41] E. M. Clarke, O. Grumberg, and M. C. Browne. Reasoning about networks with many identical finite-state processes. In *PODC*, pages 240–248, 1986.

[42] E. M. Clarke, O. Grumberg, and S. Jha. Veryfying parameterized networks using abstraction and regular languages. In *CONCUR*, pages 395–407, 1995.

[43] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Trans. Program. Lang. Syst.*, 16(5), 1994.

[44] E. M. Clarke, M. Talupur, and H. Veith. Environment abstraction for parameterized verification. In *VMCAI*, pages 126–141, 2006.

[45] B. Cook and E. Koskinen. Reasoning about nondeterminism in programs. In *PLDI*, pages 219–230, 2013.

[46] B. Cook, A. Podelski, and A. Rybalchenko. Termination proofs for systems code. In *PLDI*, pages 415–426, 2006.

[47] B. Cook, A. Podelski, and A. Rybalchenko. Proving program termination. *CACM*, 54(5):88–98, 2011.

[48] B. Cook, A. See, and F. Zuleger. Ramsey vs. lexicographic termination proving. In *TACAS*, pages 47–61, 2013.

[49] D. Cousineau, D. Doligez, L. Lamport, S. Merz, D. Ricketts, and H. Vanzetto. TLA + proofs. In *FM*, 2012.

[50] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252. ACM, 1977.

[51] G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR*, volume 6269 of *LNCS*, pages 313–327, 2010.

[52] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[53] E. A. Emerson and V. Kahlon. Model checking guarded protocols. In *LICS*, pages 361–370. IEEE, 2003.

[54] E. A. Emerson and V. Kahlon. Rapid parameterized model checking of snoopy cache coherence protocols. In *TACAS*, volume 2619 of *LNCS*, pages 144–159. Springer, 2003.

[55] E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In *POPL*, pages 85–94, 1995.

[56] E. A. Emerson and K. S. Namjoshi. On model checking for nondeterministic infinite-state systems. In *LICS*, pages 70–80. IEEE Computer Society, 1998.

[57] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao. The Daikon system for dynamic detection of likely invariants. *Science of Comp. Prog.*, 69(1):35–45, 2007.

[58] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34(2):85–107, 1997.

[59] J. Esparza, P. Ganty, and R. Majumdar. Parameterized verification of asynchronous shared-memory systems. In *CAV*, pages 124–140. Springer, 2013.

[60] Y. Fang, K. L. McMillan, A. Pnueli, and L. D. Zuck. Liveness by invisible invariants. In *FORTE*, pages 356–371, 2006.

[61] A. Farzan, M. Heizmann, J. Hoenicke, Z. Kincaid, and A. Podelski. Automated program verification. In *LATA*, pages 25–46, 2015.

[62] A. Farzan, Z. Kincaid, and A. Podelski. Inductive data flow graphs. POPL, pages 129–142, 2013.

[63] A. Farzan, Z. Kincaid, and A. Podelski. Proof spaces for unbounded parallelism. In *POPL*, pages 407–420. ACM, 2015.

[64] A. Finkel and P. Schnoebelen. Well-structured transition systems

everywhere! *Theor. Comp. Science*, 256(1–2):63–92, 2001.

[65] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[66] D. Fisman, O. Kupferman, and Y. Lustig. On verifying fault tolerance of distributed protocols. In *TACAS*, volume 4963 of *LNCS*, pages 315–331, 2008.

[67] D. Fisman and A. Pnueli. Beyond regular model checking. In *FST TCS*, pages 156–170, 2001.

[68] S. Funiak. Model checking IOA programs with TLC. Technical report, MIT, 6 2001.

[69] P. Godefroid. Using partial orders to improve automatic verification methods. In *CAV*, volume 531 of *LNCS*, pages 176–185, 1990.

[70] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *CAV*, 1997.

[71] S. Gulwani and F. Zuleger. The reachability-bound problem. In *PLDI*, pages 292–304, 2010.

[72] B. T. Hailpern and S. S. Owicki. Verifying network protocols using temporal logic. 1980.

[73] D. Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

[74] C. N. Ip and D. L. Dill. Verifying systems with replicated components in murϕ. *Formal Meth. in Sys. Design*, 14(3):273–310, 1999.

[75] M. Jaskelioff and S. Merz. Proving the correctness of Disk Paxos. Technical report, LORIA, Nancy, France, 6 2005.

[76] A. John, I. Konnov, U. Schmid, H. Veith, and J. Widder. Brief announcement: parameterized model checking of fault-tolerant distributed algorithms by abstraction. In *PODC*, pages 119–121, 2013.

[77] A. John, I. Konnov, U. Schmid, H. Veith, and J. Widder. Parameterized model checking of fault-tolerant distributed algorithms by abstraction. In *FMCAD*, pages 201–209, 2013.

[78] A. John, I. Konnov, U. Schmid, H. Veith, and J. Widder. Towards modeling and model checking fault-tolerant distributed algorithms. In *SPIN*, volume 7976 of *LNCS*, pages 209–226, 2013.

[79] T. T. Johnson and S. Mitra. A small model theorem for rectangular hybrid automata networks. In *FORTE*, pages 18–34, 2012.

[80] R. Joshi, L. Lamport, J. Matthews, S. Tasiran, M. R. Tuttle, and Y. Yu. Checking cache-coherence protocols with TLA$^+$. *Formal Methods in System Design*, 22(2):125–131, 2003.

[81] A. Kaiser, D. Kroening, and T. Wahl. Dynamic cutoff detection in parameterized concurrent programs. In *CAV*, pages 645–659, 2010.

[82] D. K. Kaynar, N. A. Lynch, R. Segala, and F. W. Vaandrager. *The Theory of Timed I/O Automata*. Synthesis Lectures on Computer Science. Morgan & Claypool Publishers, 2006.

[83] Y. Kesten, A. Pnueli, E. Shahar, and L. D. Zuck. Network invariants in action. In *CONCUR*, pages 101–115, 2002.

[84] I. Konnov, H. Veith, and J. Widder. On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. In *CONCUR*, volume 8704 of *LNCS*, pages 125–140, 2014.

[85] I. Konnov, H. Veith, and J. Widder. SMT and POR beat counter abstraction: Parameterized model checking of threshold-based distributed algorithms. In *CAV*, 2015.

[86] T. Kotek, M. Simkus, H. Veith, and F. Zuleger. Extending ALCQIO with reachability: Between finite model theory and description logic. In *LICS*, 2015.

[87] T. Kotek, H. Veith, and F. Zuleger. Monadic second order finite satisfiability and unbounded tree-width. *CoRR*, abs/1505.06622, 2015.

[88] R. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*. Princ. Univ. Press, 1995.

[89] R. P. Kurshan and K. McMillan. A structural induction theorem for processes. In *PODC*, pages 239–247. ACM, 1989.

[90] S. La Torre, P. Madhusudan, and G. Parlato. Model-checking parameterized concurrent programs using linear interfaces. In *CAV*, pages 629–644, 2010.

[91] L. Lamport. Real-time model checking is really simple. In *CHARME*, pages 162–175, 2005.

[92] L. Lamport. The PlusCal algorithm language. In *ICTAC*, volume 5684 of *LNCS*, pages 36–60, 2009.

[93] L. Lamport and S. Merz. Specifying and verifying fault-tolerant systems. In *Formal Techniques in real-time and fault-tolerant systems*, pages 41–76. Springer, 1994.

[94] J. Leroux and G. Sutre. Flat counter automata almost everywhere! In *ATVA*, volume 3707 of *LNCS*, pages 489–503, 2005.

[95] D. Lesens, N. Halbwachs, and P. Raymond. Automatic verification of parameterized linear networks of processes. In *POPL*, pages 346–357. ACM, 1997.

[96] P. Lincoln and J. Rushby. A formally verified algorithm for interactive consistency under a hybrid fault model. In *FTCS-23*, pages 402–411, 1993.

[97] N. Lynch. *Distributed Algorithms*. Morgan Kaufman, 1996.

[98] J. A. Makowsky. Algorithmic uses of the Feferman-Vaught theorem. *Ann. Pure Appl. Logic*, 126(1-3):159–213, 2004.

[99] K. McMillan and L. Zuck. Invisible invariants and abstract interpretation. In *SAS*, pages 249–262, 2011.

[100] K. L. McMillan. Parameterized verification of the FLASH cache coherence protocol by compositional model checking. In *CHARME*, 2001.

[101] K. L. McMillan. Interpolation and sat-based model checking. In *CAV*, 2003.

[102] S. Merz and H. Vanzetto. Automatic verification of TLA + proof obligations with SMT solvers. In *LPAR*, pages 289–303, 2012.

[103] F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 5(3):403–435, July 2004.

[104] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, and M. Deardeuff. How Amazon web services uses formal methods. *Communications of the ACM*, 58(4):66–73, 2015.

[105] J. W. O'Leary, M. Talupur, and M. R. Tuttle. Protocol verification using flows: An industrial experience. In *FMCAD*, pages 172–179, 2009.

[106] D. Peled. All from one, one for all: on model checking using representatives. In *CAV*, volume 697 of *LNCS*, pages 409–423, 1993.

[107] D. Peleg. *Distributed Computing: A locality-sensitive approach*. SIAM, 2000.

[108] A. Pnueli, S. Ruah, and L. D. Zuck. Automatic deductive verification with invisible invariants. In *TACAS*, pages 82–97, 2001.

[109] A. Pnueli and E. Shahar. Liveness and acceleration in parameterized verification. In *CAV*, pages 328–343, 2000.

[110] A. Pnueli, J. Xu, and L. D. Zuck. Liveness with (0, 1, ∞)-counter abstraction. In *CAV*, pages 107–122, 2002.

[111] A. Podelski and A. Rybalchenko. Transition invariants. In *LICS*, 2004.

[112] C. Popeea and A. Rybalchenko. Compositional termination proofs for multi-threaded programs. In *TACAS*, pages 237–251, 2012.

[113] J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning (2 volumes)*. Elsevier and MIT Press, 2001.

[114] R. L. Schwartz and P. M. Melliar-Smith. Temporal logic specification of distributed systems. In *ICDCS*, pages 446–454, 1981.

[115] M. Sinn, F. Zuleger, and H. Veith. A simple and scalable static analysis for bound analysis and amortized complexity analysis. In *CAV*, pages 745–761, 2014.

[116] T. K. Srikanth and S. Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2:80–94, 1987.

[117] E. W. Stark. *Foundations of a theory of specification for distributed systems*. PhD thesis, Ph. D. Thesis, MIT. LACS/TR-342, 1984.

[118] W. Steiner and B. Dutertre. Automated formal verification of the TTEthernet synchronization quality. In *NASA Formal Methods*, pages 375–390. Springer, 2011.

[119] W. Steiner, J. M. Rushby, M. Sorea, and H. Pfeifer. Model checking a fault-tolerant startup algorithm: From design exploration to exhaustive fault simulation. In *DSN*, pages 189–198, 2004.

[120] M. Talupur. *Abstraction Techniques for Parameterized Verification*. PhD thesis, CS Department, Carnegie Mellon University, 2006.

[121] M. Talupur and M. R. Tuttle. Going with the flow: Parameterized verification using message flows. In *FMCAD*, pages 1–8. IEEE, 2008.

[122] J. A. Tauber. *Verifiable compilation of I/O automata without global synchronization*. PhD thesis, MIT, 2004.

[123] G. Tel. *Introduction to distributed algorithms*. Cambridge university press, 2000.

[124] M. J. Tsai. *Code generation for the IOA language*. PhD thesis, MIT, 2002.

[125] T. Tsuchiya and A. Schiper. Verification of consensus algorithms using satisfiability solving. *Distributed Computing*, 23(5–6):341–358, 2011.

[126] A. Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri Nets 1990*, volume 483 of *LNCS*, pages 491–515. Springer, 1991.

[127] P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *CAV*, pages 88–97, 1998.

[128] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 68–80, 1989.

[129] Y. Yu, P. Manolios, and L. Lamport. Model checking TLA+ specifications. In *Correct Hardware Design and Verification Methods*, pages 54–66. Springer, 1999.

[130] F. Zuleger, S. Gulwani, M. Sinn, and H. Veith. Bound analysis of imperative programs with the size-change abstraction. In *Static Analysis*, pages 280–297. Springer, 2011.