New–Generation Symbolic Model Checkers for Verifying Asynchronous Systems

Gerald Luettgen *(PI)* Gianfranco Ciardo *(International Collaborator)*

1 Previous Research Track Record

Expertise of the PI and the international collaborator. The PI, Dr. Luettgen, has a strong background in the formal specification and verification of computer systems that exhibit asynchronous behaviour. In particular, the PI worked for one year during his PhD in the group of Prof. Cleaveland, who is one of the leading international researchers in concurrency theory and model checking. The visit was funded by a doctoral grant of the German Academic Exchange Service (DAAD) and laid the basis of a PhD thesis that was selected by the German Computer Society as one of the finalists of the 1998 Distinguished Dissertation Award. The work comprised the development of novel semantic theories of priority and real–time in distributed systems, their implementation in an automated design and verification tool, and the conduct of case studies using model checking [4, 19]. The PI has also considerable experience in algorithms for state–space generation. In particular, he co–developed a novel state–space generator that allows for the compositional on–the–fly minimisation of state spaces of asynchronous systems [22].

The international collaborator Prof. Ciardo is a full professor at the College of William & Mary in Virginia and a co–author of this research proposal. He is an internationally recognised expert in the performability evaluation of fault–tolerant and distributed computer systems. His particular area of expertise is in the logical and stochastic analysis of concurrent software, including model checking and Kronecker–based techniques. The collaborator has devised SMART [13], a modelling tool for the logical and timing analysis of complex systems, which is built around the specification formalism Petri nets [31] and will be used in the context of the proposed project. Prof. Ciardo is also an Associate Editor of the renowned journal IEEE Transactions on Software Engineering.

Collaboration track record. During the PI's employment as Staff Scientist at the Institute for Computer Applications in Science and Engineering at NASA Langley Research Center in Virginia, the PI and the collaborator joined forces to develop novel decision–diagram–based algorithms for efficiently generating the huge state spaces underlying *event–based asynchronous systems*, such as distributed embedded software.

In contrast to all related work, the PI and the collaborator have proposed a fundamentally novel way to address the traditional space and time limitations of state–space–generation algorithms. This approach employs *Multi–valued Decision Diagrams* (MDDs) [25] for encoding state spaces, paired with a *Kronecker encoding* of a system's underlying next–state function. While being a relatively straightforward extension of *Binary Decision Diagrams* (BDDs) [7], MDDs are essential to enable our Kronecker representation. The combination of these two ingredients paves the way for devising model–checking algorithms that allow one to rigorously exploit the structural properties of the asynchronous systems under study. In essence, the key effect is to enable *local* manipulations of MDDs by taking into account the local effect of the firing of events. This is in contrast to traditional approaches that use blunt monolithic applications of a system's next–state function, which is normally encoded as one huge decision diagram. Experiments with prototypical sequential implementations of algorithms for state–space generation (but not model checking), which incorporate our ideas and have been recognised by the Petri net and TACAS communities [11, 12], have shown run–time and memory–efficiency improvements of up to four orders and three orders of magnitude, respectively, when compared to state–space generation within the state–of–the–art model checker NuSMV [16]; this model checker is implemented around optimised BDD techniques and recently integrates also techniques based on SAT solving [5]. In addition to the dining philosophers model taken from the NuSMV distribution, our comparison considered models of a slotted ring network [33], a round–robin mutual exclusion protocol [22], and a flexible manufacturing system [11]. Each model was parameterised by the number of its subcomponents or other quantities affecting its state–space size, and the observed performance gaps kept increasing as models grew.

This previous work of the PI and the collaborator builds the basis for the proposed research. It provides a foundation for devising new–generation MDD–based model checkers for the temporal logic CTL [17]. The capability of performing localised MDD manipulations will also enable the development of *parallel* model–checking algorithms along very different paths than those explored in the literature. This promises significant speed–ups for both shared– and distributed–memory implementations in a field, where relatively little has been achieved to date. The collaborator has already secured funding for the jointly proposed line of research within the very competitive Information Technology Research Programme of the National Science Foundation (NSF) in the United States. While the NSF supports the collaborator by funding two post–graduate students at William & Mary for three years, it cannot support the PI in York according to its policies. The present EPSRC research proposal thus seeks funding for the efforts in York within this international collaboration.

Institutional expertise and support. The PI has just joined the Computer Science Department at the University of York, which has attracted the highest possible rating (5*) in the most recent national Research Assessment Exercise. The Department has several internationally renowned research groups with expertise in the fields of

formal methods and verification, in particular regarding their application to High–Integrity Systems Engineering, Human–Computer Interaction, Programming Languages and Systems, and Real–Time Systems. Current research in the Department is funded by grants and contracts totalling £5 million from the EPSRC, the European Commission, other government departments and industry. The Department's excellent relations with industry have led to its designation as a University Technology Centre by Rolls–Royce and as a Centre of Excellence by British Aerospace (BAe), as well as to the establishment of a Dependable Computing Systems Centre.

2 Description of Proposed Research

2.1 Background

A key component of today's technology–driven society is the *software for distributed embedded systems*; examples include communications and electronic–commerce protocols as well as avionics and automotive systems. Despite their growing presence, distributed embedded software is difficult and expensive to design and debug. The main challenge lies in the inherent *asynchrony* of distributed software, which may result in subtle and often unanticipated interactions between system components. Practice has shown that, despite extensive testing, errors still arise with sometimes devastating consequences, either financially or for personal safety.

A mathematical approach complementing testing is *formal verification*. Research in this field over the past two decades has led to the emergence of fully–automated verification techniques, such as *temporal–logic model checking* [17, 18, 34]. In a nutshell, temporal–logic model checking is a decision procedure for checking finite state spaces of systems against behavioural specifications given as temporal–logic formulas. To make model checking cope with the large state spaces of many real–world systems, symbolic state–space representations based on BDDs [7] have been adopted in industrial–strength model checkers [10]. This triggered the success of model checking in the hardware industry [21], as the behaviour of synchronous digital circuits can be encoded naturally into BDDs.

Despite the many reported advantages of BDD–based model checking [9] for finding subtle errors in complex systems, it is very sparsely employed in the development of software for distributed embedded systems. Indeed, it faces serious problems when applied to *asynchronous* systems. The reason for this is the inherent high complexity of distributed systems, since their state spaces tend to explode in the number of system components. This does not only lead to BDDs that are too large to fit into the memory of a single workstation, but it also requires a fair amount of time for constructing the state spaces of interest [33]. In practice, many engineers use model checking as an advanced debugging technique and expect feedback within minutes, not hours. Hence, for asynchronous software, state–space generation and exploration become time–bound problems, in addition to memory–bound problems.

Related Work. Of most direct relevance to this project is work on sequential and parallel symbolic model checking based on BDDs [23, 26, 29]. A BDD is a compact representation of a full Boolean decision tree over a given number of variables, obtained by merging common subtrees. It can be exponentially more compact than its corresponding decision tree, but the degree of compactness depends on the chosen variable ordering [7]. Efficient implementations of BDDs, often in form of C++ packages [40], employ two hash tables: a *unique table* to store and retrieve BDD nodes, and a *cache* to avoid evaluating an operation on a given set of BDD nodes more than once.

Important applications of BDDs are in state–space generation and temporal–logic model checking [9, 29], especially for the temporal logic CTL [17]. To cope with the ever increasing complexity of real–world systems, several approaches have been studied to further improve the efficiency of BDD–based algorithms. For improving time efficiency, breadth–first BDD–manipulation algorithms [1] have been explored and compared against the traditional depth–first ones. However, the results show no significant speed–ups, although breadth–first algorithms lead to more regular access patterns of unique tables and caches. For improving space efficiency, a fair amount of work has concentrated on choosing appropriate variable orders [20] and on re–ordering variables on–the–fly [37].

Several avenues for parallelising model checkers have been investigated in the literature, in particular for explicit–state (rather than symbolic) model checkers [3]. Due to space constraints we focus on the parallelisation of BDD–based algorithms, which is particularly challenging since the traditional depth–first recursive BDD algorithms are inherently sequential [23]. Although several efforts have been made to parallelise algorithms for generating and manipulating BDDs on shared– and distributed–memory architectures, thereby utilising the larger memory and higher computation power available on those machines, little has been achieved. Indeed, previous work on distributed–memory BDD algorithms stresses the ability of using the overall amount of memory on a network of workstations, but does not result in meaningful speed–ups when compared to single–processor implementations, unless the latter start using virtual memory [24]. In [35], BDD nodes are considered in a breadth–first order to improve memory access patterns and to reduce communication; however, four–processor execution is up to three times slower than single–processor execution. Similarly, [39] shows speed–ups only when the sequential execution cannot fit in a machine's main memory. In [30], a small speed increase is achieved for hardware circuits; the solution time on eight processors is 7% faster at best, but 176% slower at worst, when compared to the same code on a single processor. Better speed–ups have instead been demonstrated for shared–memory BDD algorithms. For example in [26], a speed–up factor of 10 is achieved using 15 of the 16 processors of an Encore Multimax.

2.2 Research Programme and Methodology

2.2.1 Aims and Objectives. The key aim of this project is to develop novel, mathematically well–founded and practically useful symbolic model–checking algorithms for verifying event–based asynchronous systems. The overall objective is to achieve performance improvements of several orders of magnitudes when compared to internationally leading model checkers, thereby removing the key obstacle behind the current poor uptake of model–checking technology in the industrial development of distributed embedded–systems software. Specific objectives, with an emphasis on the research to be conducted in York, include the following:

• *MDD–based model–checking algorithms.* We propose to devise time– and space–efficient sequential symbolic model–checking algorithms for asynchronous systems. The novelty of these algorithms will be the use of MDDs and Kronecker operators instead of the much more common BDDs and, most importantly, the exploitation of structural aspects of asynchronous–systems models as well as of algorithmic aspects affecting the order in which MDD nodes are explored. In particular, York will investigate the structural aspects of *event locality* and *partitioning*, explore the algorithmic aspect of *iteration control*, and conduct formal correctness proofs for the devised algorithms.

• *Parallelisations of these algorithms.* We will investigate parallelisations of the above algorithms for shared– and distributed–memory architectures. Unlike previous approaches to parallelise BDD operations mentioned in the background section, we intend to find *parallelism at the event level* by exploiting event locality. We expect that this will significantly speed–up state–space exploration algorithms while utilising the larger memory available on parallel machines, in particular PC/workstation clusters. In achieving this objective, the two most important contributions of York will be (i) identifying how to best achieve high levels of parallelism, given the flexibilities provided by event locality and iteration control, and (ii) conducting correctness proofs for the devised parallel algorithms.

• *Implementation and validation.* We will implement all of the above algorithms in form of C++ packages, integrate them in the tool SMART [13], and additionally make them web–accessible for remote execution. Validation of our work will involve benchmarking as well as the conduct of two practically relevant case studies in the analysis of human–computer–interaction and the verification of parallel algorithms. While the implementation effort and the validation effort will be shared between York and William & Mary.

2.2.2 Methodology. We plan to follow up our original ideas for overcoming the space and time limitations of existing model–checking algorithms for the class of event–based asynchronous systems. In the following we address the key novelties of our envisioned model–checking technology, which centre around the concepts of *Multi–valued Decision Diagrams* and *Boolean Kronecker encodings*, *event locality*, and *iteration control*. It should be noted that the ability for significant advances in the field does not lie within each concept in isolation but within the interplay between these concepts. Together the concepts permit the systematic exploitation of the structural and behavioural properties of asynchronous systems. In particular, they allow for devising data structures that (i) facilitate the *compact* storage of state spaces, thereby enabling space efficiency, and (ii) ensure that frequent operations on state spaces involve only *local* manipulations of these data structures, thereby enabling time efficiency. In particular, (ii) is not achieved for the class of asynchronous systems by any related work.

Multi–valued Decision Diagrams and Boolean Kronecker encodings for storing state sets and next–state functions. Event–based asynchronous–systems models are usually either composed of some number K of subsystems, as is the case for models of embedded software distributed over K sites, or can be *partitioned* into K subsystems, as is the case for Petri net models [14]. Such discrete–state systems give rise to global states that can be represented as a K–tuple of local states. Assuming finite state spaces, we can enumerate each local state space S_k using traditional state–space generation techniques and identify it with an initial interval of natural numbers. Hence, a global state corresponds to a K–tuple of natural numbers, and a set of states S can be represented by its characteristic function mapping the set $S_1 \times \cdots \times S_K$ to $\{0,1\}$, which in turn can be encoded by an MDD [25].

The next–state function N of an event–based asynchronous system model determines, given the global state the system is currently in and an event α, the state that the system will enter upon the occurrence of α. This function is usually encoded as a $2K$–level decision diagram over variables $(x_1, y_1, \ldots, x_K, y_K)$, where x_k and y_k refer to the k–th state component before and after the firing of some event, respectively. We depart again from related work and encode the next–state function N not as an MDD but as K Boolean matrices. To do so, we partition N into one next–state function N_α per system event α, where $N_\alpha(i_1, \ldots, i_K)$ describes the states reachable from state $(i_1, \ldots, i_K)$ when event α occurs. This form of *disjunctive partitioning* is applicable to systems expressed in almost all high–level languages employed for asynchronous–systems modelling, including Petri nets, process algebras, and interleaving–based concurrency models, and has long been considered effective for asynchronous systems [8]. Note that the functions N_α can be extracted automatically from models in high–level languages [14]. We use a further decomposition of the next–state functions N_α along a different axis, by expressing the effect of an event on each submodel (or level of the MDD). The result is a very efficient representation of N by the Boolean Kronecker expression $\sum_{\alpha\in E} N_\alpha = \sum_{\alpha\in E} \bigotimes_{1\le k\le K} N_{k,\alpha}$, i.e., $(j_1, \ldots, j_K) \in N(i_1, \ldots, i_K) \iff \exists \alpha \in E\, \forall k \in \{1, \ldots, K\}.\, j_k \in N_{\alpha,k}(i_k)$. In our work, each $N_{k,\alpha}$ will then be represented as a $|S_k| \times |S_k|$–matrix over $\{0,1\}$, which results in a very compact encoding of N on which calculations can be performed efficiently.

Event locality. Other key improvements we propose emerge from the inherent *event locality* of asynchronous event–driven systems. In such systems, the occurrence of an event usually only changes a few components of the global state vector. Due to our mapping of subsystems to MDD levels, we can exploit event locality to explore MDDs only between the highest and the lowest level affected by each event. This range of levels can be extracted automatically from the high–level description of the system prior to state–space generation/exploration. To benefit from this observation when exploring new states reached by the firing of an event, we access MDD nodes directly at the highest level affected by this event, rather than always starting MDD operations at the root, as traditional approaches do. Moreover, operations on MDDs — in particular the operation corresponding to the union of state sets — can update nodes *in–place* rather than returning their results by creating new nodes, as is done in previous work.

Iteration control. The flexibility gained by event locality particularly concerns the order in which MDD nodes and system events may be considered by fixed–point operations, which is the key technology behind model checking [18, 29]. In our initial experiments with state–space generation [11, 12], we have partitioned the set of system events E into K classes, $E_1, \ldots, E_K$, where E_k contains all events affecting level k and possibly lower levels. Then, we explore the MDD nodes not as is normally done in depth–first or breadth–first fashion starting at the root, but in a novel way to which we refer as *saturation* [12]. Given the MDD encoding of the initial system states, we consider each node at a given level, starting from the lowest level and moving up, and repeatedly fire all events affecting this level until no more states are discovered. This saturation can cause the creation of new nodes at lower levels, which are then saturated recursively before completing the saturation of the nodes above them. The resulting algorithm has many desirable properties: the unique table needs to store only the saturated nodes, and these are likely to remain useful throughout the execution; the operation caches are used in a much more localised way, thus their memory requirements are greatly reduced; finally, an event is explored on nodes whose descendants are saturated, whence again increasing the likelihood of finding new states early. Saturation is innovative in that it has no concept of global fixed–point iteration; state–space generation ends when the root MDD node is reached and saturated.

2.2.3 Programme of Work. The following focuses on those parts of the research agenda for which EPSRC funding is sought. The contributions of York within the larger context of the international project concern issues involving the exploitation of semantic properties of asynchronous systems, model–checking techniques and proofs of correctness, all of which are within the expertise of the PI. These issues complement, but are not always orthogonal to, the algorithmic and tool–integration issues to be investigated by the collaborator under his NSF grant.

Phase I: Sequential MDD–based model checking. The aim of the first phase is to develop, implement and analyse an efficient sequential symbolic model–checker for asynchronous–systems models based on the above ideas. The focus of the work in York is on investigating the following specific questions, all of which require a systematic exploitation of the semantic properties of the high–level formalisms in which system models are expressed. We are most interested in the high–level formalism of Petri nets [31], due to its widespread use and since it is already implemented in the tool SMART [13], in which we are planning to integrate our algorithms.

• *Can one use iteration control strategies for CTL model–checking?* We have clearly shown in [12] that saturation excels at state–space generation. This directly lends itself to evaluating the AG operator of the temporal logic CTL [17]. A formula EGϕ, however, requires us to restrict the exploration only to paths along which property ϕ holds at all times. This makes for a quite difficult adaptation of saturation, because a constant intersection of the newly found reachable states with the set of states satisfying ϕ is required. While this issue is currently investigated by the collaborator in the context of his NSF grant, the PI envisions that a structural analysis of the Petri net under consideration might be the best way forward. Such an analysis could determine which events may affect ϕ, in order to apply saturation only for those events that do not affect ϕ.

• *How to obtain a good partition?* Each MDD–level in our setting stores the state information with respect to one subsystem of the system under consideration. There is usually considerable choice when splitting a system into subsystems. For example, systems may be specified in Petri nets [31], for which it is possible to derive automatically the finest partition respecting the requirements of Boolean Kronecker operators. However, the finest partition might not be the most beneficial one for our state–space exploration algorithm, in terms of both space–efficiency and time–efficiency, as our initial studies have shown [11]. Thus, we propose to investigate heuristics to coarsen the finest partition, with the goal of reducing space and time requirements. Again, we expect structural model information, e.g., place–invariants in Petri nets, to provide the basis for such heuristics.

• *How to formally verify our algorithms?* Due to the subtleties of the issues involved, the PI in York plans to formally prove the correctness of our algorithms with the help of a theorem prover, such as PVS [32], with which he is familiar. This requires building a library for reasoning about MDDs and devising proof tactics that can discharge proofs of simple statements on MDDs automatically. To the best of our knowledge, such a library has not been developed before. The MDD library will almost certainly yield useful insights into the properties of our model–checking algorithms, which in turn might be exploited to improve our model checkers further. In addition, this approach is likely to provide clues on how to combine new approaches to software verification based on theorem proving into our algorithms, too.

The collaborator at William & Mary will use our results as a basis for implementing an efficient sequential model checker for verifying asynchronous systems modelled via Petri nets. This is currently work in progress, which has already shown first very promising results [15] recognised by the computer–aided verification community, but requires further research to be conducted by the collaborator regarding two aspects. First, BDDs sizes are known to depend on the underlying variable order [29]. In our framework, however, the order of variables also depends on the order of partitions. Thus, any heuristics based on variable ordering within MDD implementations must also consider partition ordering. Second, while state–space generation builds a single MDD, model checking operates on multiple sets of states, each encoded as an MDD over the same set of variables. For efficiency purposes, nodes must be shared between different MDDs, just as in BDD approaches [2]. However, the existing BDD approaches need to be revised for our framework, since our algorithms will rely on updating nodes on–the–fly.

The implementation of our novel model checker will be in the form of C++ packages. These will be integrated by the collaborator in the SMART tool [13], a modelling tool for the logical and timing analysis of complex systems, which is written in C++ and was developed at the College of William & Mary under direction of the collaborator. The architecture of SMART has a clear separation between front–end (user–visible interface and the Petri net language) and back–end (currently Markov solvers and discrete–event simulators). This means that the integration of model checkers can be done with relatively little effort, and any other technique requiring state–space generation as a preliminary step can immediately make use of the new techniques we will provide.

The performance of our novel sequential model checker will be analysed both in York and at William & Mary by applying it to several examples taken from the literature and from the distribution of existing model checkers. In particular, we will compare our model checker to the current state–of–the–art model checker NuSMV [16] which implements BDD–based techniques as well as novel techniques based on SAT solving [5]. In addition, this algorithm will not only be disseminated in form of C++ packages but also be made web–accessible for remote execution, which will allow us to gather further performance statistics that will be used to fine–tune our implementations.

Phases II & III: Shared–memory and distributed–memory algorithms. As sizes of MDDs grow very rapidly when studying asynchronous–systems models, parallel algorithms for MDD–manipulations might be the answer for being able to obtain verification results at all and for providing them in a timely manner.

Parallelisation for shared–memory architectures. Here, two research questions will be central. Their investigation will be led by York as they require good skills in formal semantics and in conducting correctness proofs.

• *What sources of parallelism should be exploited?* Our sequential approach lends itself to both event–based parallelism, where each event is explored by a single thread over entire MDDs, and level–based parallelism, where all events are explored by a single thread over a given level. It is possible that the best option will be a mixture of the two. Neither kind of parallelisation has been considered for symbolic state–space generation/model–checking before.

• *How to maintain cache consistency?* Regardless of the parallelisation choice, it is certain that the operation caches will be examined by many threads. Since cache look–up is an extremely frequent operation, it will be imperative to use progressive cache access protocols that do not result in large run–time penalties. However, such protocols are notoriously hard to implement correctly and any resulting algorithm should be formally proved correct by using our theorem–prover library developed in the first project phase.

Other important aspects, mostly aspects of an algorithmic nature, will be addressed by the collaborator at William & Mary. These include the question of how to minimise idle time of processors. With event–based parallelism, threads will require locks on individual MDD nodes to ensure exclusive access. Our algorithms must exploit the large number of MDD nodes to their advantage, thereby reducing the likelihood of threads having to wait at locked nodes.

Parallelisation for distributed–memory architectures. Our algorithm design will focus on two goals: achieving parallelism and reducing communication overhead. Event locality will play a key role for achieving both.

• *How to achieve high, scalable levels of parallelism?* When partitioning MDD levels into as many contiguous ranges as there are available processors, event locality will allow us to trigger event exploration at the required local levels. Thus, events in different ranges will be naturally explored in parallel, without the obvious bottleneck of having to start all work at the roots of MDDs. Alternatively, we will investigate the allocation of individual nodes to processors, with the goal of keeping the range of most events within a single processor.

• *How to achieve memory and load balance?* With a partition of levels over processors, memory and load unbalance can be corrected by shifting the border between ranges. With the finer granularity achieved by assigning nodes to processors, balancing should be easier. In the context of partitioning the levels or nodes over the available processors, one may want to minimise the number of arcs connecting nodes assigned to different processors.

• *How to best manage message passing?* In most MDD operations, work requests are propagated towards the MDD leafs, while results are propagated towards the MDD roots. Our ideas for parallelisation require us to pass messages between different processors. While the standard technique of batching messages allows for lowering communication overhead, it will have to be tuned for our application in order to avoid increasing processor idle time.

While York will be taking the lead regarding the first question, William & Mary will focus on the other questions. However, the exact issues to be addressed and the distribution of work can only be finalised after we will have

reflected on our experiences gained in Phases I and II. We will do so in the project meeting at the start of Phase III.

Implementation and performance validation. Implementation and performance validation of our shared–memory and distributed–memory algorithms will be a shared effort between York and William & Mary, in particular for distributed–memory architectures. The shared–memory algorithms will initially be developed and evaluated on two machines with together 28 Ultra Sparc III processors and 68GB of RAM, which are available to us in York. The distributed algorithms will be implemented using MPI [38] and run on PC/workstation–clusters. One such large cluster is made available to this project via the White Rose Grid (*www.whiterose.ac.uk/HPDGrid.cfm*), to which the University of York has access, and another one via the collaborator, whose department is a member of the College of William & Mary Computational Science Cluster (*www.compsci.wm.edu*). The White Rose Grid offers several Beowulf systems with up to 256 nodes, while the William & Mary Science Cluster consists of three Beowulf–style subsystems, two of which are Pentium–based and have 96 processors altogether, while the third subsystem contains four Sun multiprocessor machines. We are planning to use a cluster within the White Rose Grid for prototyping and the William & Mary cluster for gathering detailed performance statistics of our model checkers. The reason is that the latter allows us to fully control the operating environment. All algorithms will be made available in the form of C++ packages, which allows for their easy integration into existing formal verification tools [16], and will also be integrated into the collaborator's tool SMART [13].

To assess the efficiency of our parallel model checkers, they will initially be applied to the same example systems as the sequential algorithms. Moreover, we intend to conduct an in–depth performance comparison to other popular model checkers, including NuSMV [16], which is a nontrivial task due to the following observation. The performance of symbolic model checkers is usually measured by applying them to the well–established ISCAS benchmarks [6], whose examples are taken from digital circuit design. These circuits have a synchronous semantics, whence the ISCAS benchmarks are not suitable to validate our model–checking algorithms that are targeted towards verifying event–based asynchronous systems. Indeed, the current literature does not provide any suitable benchmark for such systems. We thus propose to devise a new benchmark by systematically collecting a variety of asynchronous example systems that have been studied in the diverse literature on formal verification and concurrency theory. This benchmark will be well–documented, disseminated via the Internet and used to evaluate our algorithms.

Phase IV: Case studies. Both PI and collaborator will further apply our algorithms in the context of two medium–scale case studies; this will testify to the practical relevance of our work.

The first case study, to be conducted in York and demonstrating our novel algorithms in practice, will draw on the PI's experience in the analysis of flight–guidance systems, which aims at revealing potential sources of *mode confusion* in aircraft cockpits [27]. Mode confusion arises when the pilot's perception of the current state of the plane (in terms of altitude, heading, etc.) does not correspond to the state that the digital flight deck is in. Human–factors experts have identified mode confusion as significant contributor to aviation accidents and incidents. As Staff Scientist at a research institute at NASA, the PI has successfully analysed synchronous components of an idealised flight guidance with regard to mode–confusion properties. These were specified as temporal–logic formulas [28] and verified using the BDD–based model checker SMV [29]. We propose to investigate other challenging mode confusion properties that involve the gathering and display of mode information in the cockpit. Many properties of interest to human–factors experts can only be thoroughly analysed when also several parts of the operating environment of flight–guidance and flight–control systems, such as switching panels, displays, and sensor and actuator behaviour, are taken into account. This will lead to event–based asynchronous models with enormous state spaces that can either not be handled by existing symbolic model checkers at all or not in a time–efficient manner. This study will thus be an excellent candidate for validating the robustness of our novel algorithms in practice.

A second case study, aimed at verifying a shared–memory algorithm for *parallel B–tree manipulation*, will be conducted at William & Mary with funding from the NSF. We shall briefly mention it here to highlight the versatility of our approach. B–trees are a popular data structure to index very large data sets. In distributed database environments, concurrent access to B–trees is often achieved by using locks, thus resulting in performance bottlenecks. Novel approaches based on *speculative computing* [36] do not prevent concurrent access to a node, but record modification counts of nodes for deciding whether an operation was safe. Establishing the correctness of such speculative approaches is hard since all interleavings of multiple writer/reader executions must be taken into account. We believe that our parallel model checkers will enable the verification of the algorithm for realistic operating environments.

2.2.4 Workplan. Our workplan comprises four phases: (i) developing and evaluating model checkers for sequential architectures; (ii) developing and evaluating model checkers for shared–memory architectures; (iii) developing and evaluating model checkers for distributed–memory architectures; (iv) conducting an in–depth comparison of the devised model checkers on the basis of case studies. Every phase starts with a joint project meeting between the PI and the international collaborator. The first three phases will each conclude with a finished product, namely a model checker encapsulated in a C++ package, integrated into the tool SMART and additionally executable via the World–Wide Web, together with an analysis of the model checker's correctness and its performance in light of related work. Our detailed plans can be found in the attached diagrammatic workplan, which also shows the distribution of

work between York and William & Mary. This workplan is compatible with the collaborator's workplan as stated in the partner research proposal that has already been selected for funding by the US National Science Foundation. The UK effort focuses on those aspects that require expertise in semantics, proofs of correctness and model checking.

2.3 Relevance to Beneficiaries and to the Collaborator

This project will significantly enhance the applicability of state–exploration techniques and symbolic model checking for the verification and analysis of asynchronous systems, such as distributed embedded software. It will benefit academic and industrial researchers, as well as developers and users of formal verification and design tools.

From a researcher's point of view, the proposed project will take an important step forward in the challenging field of automated software verification, as much of today's software is inherently asynchronous. By bringing to light and carefully analysing the issues involved in efficiently model–checking asynchronous systems, the project is likely to raise new questions, of both practical and theoretical nature, that should be of interest to the wider Computer Science and Engineering communities. From a tool builder's point of view, our C++ packages will facilitate the integration of our model checkers in existing verification and validation tools and in industrial design tools.

The users of verification technology, such as engineers designing embedded–systems software, will benefit from the integration of our algorithms in the collaborator's modelling and verification tool SMART, which is in the public domain and ready to use. The successful conduct of this research project is thus of high relevance to the collaborator Prof. Ciardo and the international users of SMART, too.

2.4 Dissemination and Exploitation

The dissemination of our research results will proceed along two routes. The first, more classical route will take the project reports as a basis for presentations at international conferences and publications in renowned journals. The conferences of interest include those traditionally dedicated to formal verification, such as the Intl. Conf. on Computer Aided Verification, the Intl. Conf. on Foundations of Tools and Algorithms for the Construction and Analysis of Systems, the IEEE Intl. Conf. on Computer–Aided Design, the Intl. Workshop on Parallel and Distributed Model Checking, and the Intl. Conf. on Formal Methods in Computer–Aided Design. Targeted journals include Formal Aspects of Computing, Formal Methods in Systems Design, Software Tools for Technology Transfer, and the Journal of the ACM. We also intend to present our results at broadly–targeted software engineering venues, such as the Intl. Conf. on Software Engineering, where our proposed research can make a faster practical impact.

Second, the developed model–checking packages will be made web–accessible for remote execution via the Internet, thereby promoting their uptake and dissemination in the wider formal–verification and embedded–systems communities. This dissemination route is in addition to the traditional route via C++ packages and also aimed at providing us with feedback regarding our algorithms' performance in practice.

A successful conclusion of this project would establish a solid case for the commercial exploitation of our model–checking algorithms; the commercial potential will be assessed as part of the final project report.

2.5 Justification of Resources

The proposed research is embedded in a larger bilateral research project. While the collaborator's research share at William & Mary is already funded by the NSF in the US, the PI is herewith seeking support for the UK activities within this international collaboration. These activities entail a substantial amount of research which the PI is not in a position to undertake by himself. Funding is therefore requested for a Post Doctoral RA in York for the full duration of the project. Support is also requested for the collaborator, Prof. G. Ciardo, to visit the PI annually as a visiting fellow, twice for three weeks and once for two weeks. In addition to covering his travel expenses, a stipend will be necessary as US salaries are only nine–months salaries that do not cover research consultancies. It shall be emphasised that these costs are *not* covered by the collaborator's NSF grant.

To implement the proposed model checkers, the RA will need full–time access to a dedicated computer, for which funds are also requested. Due to the high volume of programming involved, a state–of–the–art dual processor PC with at least 2GB main memory is needed; all required software is in the public domain. Access to the high–performance computers of the White Rose Grid is available to us free of charge. However, their configuration for and maintenance within this project is a complex task that cannot be mastered by the RA, but requires a trained professional. Funding is thus sought to support a technician part–time, for an average of one day per week throughout the duration of the project. The usage, configuration and maintenance expenses for the Computational Science Cluster at William & Mary are covered by the NSF.

Funds are requested for the RA to visit the collaborator at William & Mary twice, for three weeks each in the first and second project year. These visits will enable her/him to receive training on the SMART tool and on the usage of and the programming on William & Mary's workstation cluster. Additionally, the PI will need to travel to William & Mary twice, for one week each, in order to coordinate the project with the collaborator; other travel of the PI to the collaborator is already funded under the collaborator's NSF grant. Finally, travel support is requested for attending three international conferences each year, in order to facilitate the dissemination of our research results.

References

[1] P. Ashar and M. Cheong. Efficient breadth–first manipulation of binary decision diagrams. In *ICCAD '94*, pp. 622–627. IEEE Comp. Soc. Press, 1994.

[2] H. Babu and T. Sasao. Shared multi–terminal binary decision diagrams for multiple–output functions. *IEICE Trans. on Fundamentals of Electronics Communications and Computer Sciences*, E81–A(12):2545–2553, 1998.

[3] H. Barringer and C.P. Inggs. Effective state exploration for model checking on a shared memory architecture. *ENTCS*, 68(4), 2002.

[4] G. Bhat, R. Cleaveland, and G. Luettgen. A practical approach to implementing real–time semantics. *Annals of Softw. Eng.*, 7:127–155, 1999.

[5] A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *DAC '99*, pp. 317–320. ACM Press, 1999.

[6] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits. In *ISCAS '85*, pp. 695–698. IEEE Comp. Soc. Press, 1985.

[7] R.E. Bryant. Graph–based algorithms for Boolean function manipulation. *IEEE Trans. on Computers*, 35(8):677–691, 1986.

[8] J.R. Burch, E.M. Clarke, and D.E. Long. Symbolic model checking with partitioned transition relations. In *Intl. Conf. on Very Large Scale Integration*, pp. 49–58. North–Holland, 1991.

[9] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inform. and Comp.*, 98(2):142–170, 1992.

[10] Cadence, Inc. FormalCheck. www.cadence.com.

[11] G. Ciardo, G. Luettgen, and R. Siminiceanu. Efficient symbolic state–space construction for asynchronous systems. In *ICATPN 2000*, LNCS, pp. 103–122, 2000.

[12] G. Ciardo, G. Luettgen, and R. Siminiceanu. Saturation: An efficient iteration strategy for symbolic state–space generation. In *TACAS 2001*, vol. 2031 of *LNCS*, pp. 328–342, 2001.

[13] G. Ciardo and A.S. Miner. SMART: Simulation and Markovian Analyzer for Reliability and Timing. In *IPDS '96*, p. 60. IEEE Comp. Soc. Press, 1996.

[14] G. Ciardo and A.S. Miner. A data structure for the efficient Kronecker solution of GSPNs. In *PNPM '99*, pp. 22–31. IEEE Comp. Soc. Press, 1999.

[15] G. Ciardo and R. Siminiceanu. Structural symbolic CTL model checking. In *CAV 2003*, vol. 2725 of *LNCS*, pp. 40–53, 2003.

[16] A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A new Symbolic Model Verifier. In *CAV '99*, vol. 1633 of *LNCS*, pp. 495–499, 1999.

[17] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite–state concurrent systems using temporal logic specifications. *TOPLAS*, 8(2):244–263, 1986.

[18] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[19] R. Cleaveland, V. Natarajan, S. Sims, and G. Luettgen. Modeling and verifying distributed systems using priorities: A case study. *Software–Concepts and Tools*, 17(2):50–62, 1996.

[20] M. Fujita, H. Fujisawa, and Y. Matsunaga. Variable ordering algorithms for ordered binary decision diagrams and their evaluation. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 12(1):6–12, 1993.

[21] R. Goering. Model checking expands verification's scope. *EE Times*, vol. 939, Feb. 3, 1997.

[22] S. Graf, B. Steffen, and G. Luettgen. Compositional minimisation of finite state systems using interface specifications. *Formal Aspects of Computing*, 8:607–616, 1996.

[23] O. Grumberg, T. Heyman, and A. Schuster. Distributed symbolic model checking for μ-calculus. In *CAV' 01*, vol. 2102 of *LNCS*, pp. 350–362, 2001.

[24] T. Heyman, D. Geist, O. Grumberg, and A. Schuster. A scalable parallel algorithm for reachability analysis of very large circuits. *FMSD*, 21(3):317–338, 2002.

[25] T. Kam, T. Villa, R.K. Brayton, and A. Sangiovanni-Vincentelli. Multi–valued decision diagrams: Theory and applications. *Multiple–Valued Logic*, 4(1–2):9–62, 1998.

[26] S. Kimura and E.M. Clarke. A parallel algorithm for constructing binary decision diagrams. In *ICCD '90*, pp. 220–223. IEEE Comp. Soc. Press, 1990.

[27] N.G. Leveson, L.D. Pinnel, S.D. Sandys, S. Koga, and J.D. Reese. Analyzing software specifications for mode confusion potential. In *Workshop on Human Error and System Development*, 1997.

[28] G. Luettgen and V. Carreño. Analyzing mode confusion via model checking. In *SPIN '99*, vol. 1680 of *LNCS*, pp. 120–135, 1999.

[29] K.L. McMillan. *Symbolic Model Checking*. Kluwer, 1993.

[30] K. Milvang-Jensen and A.J. Hu. BDDNOW: A parallel BDD package. In *FMCAD '98*, vol. 1522 of *LNCS*, pp. 501–512, 1998.

[31] T. Murata. Petri nets: Properties, analysis and applications. *Proc. of the IEEE*, 77(4):541–579, 1989.

[32] S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal verification for fault–tolerant systems: Prolegomena to the design of PVS. *IEEE Trans. on Softw. Eng.*, 21(2):107–125, 1995.

[33] E. Pastor, O. Roig, J. Cortadella, and R.M. Badia. Petri net analysis using Boolean manipulation. In *ICATPN '94*, vol. 815 of *LNCS*, pp. 416–435, 1994.

[34] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Intl. Symp. on Programming*, vol. 137 of *LNCS*, pp. 337–351, 1982.

[35] R.K. Ranjan, J.V. Sanghavi, R.K. Brayton, and A. Sangiovanni-Vincentelli. Binary decision diagrams on networks of workstations. In *ICCD '96*, pp. 358–364. IEEE Comp. Soc. Press, 1996.

[36] P.K. Reddy and M. Kitsuregawa. Improving performance in distributed database systems using speculative transaction processing. In *European Parallel and Distributed Systems*. ACTA Press, 1998.

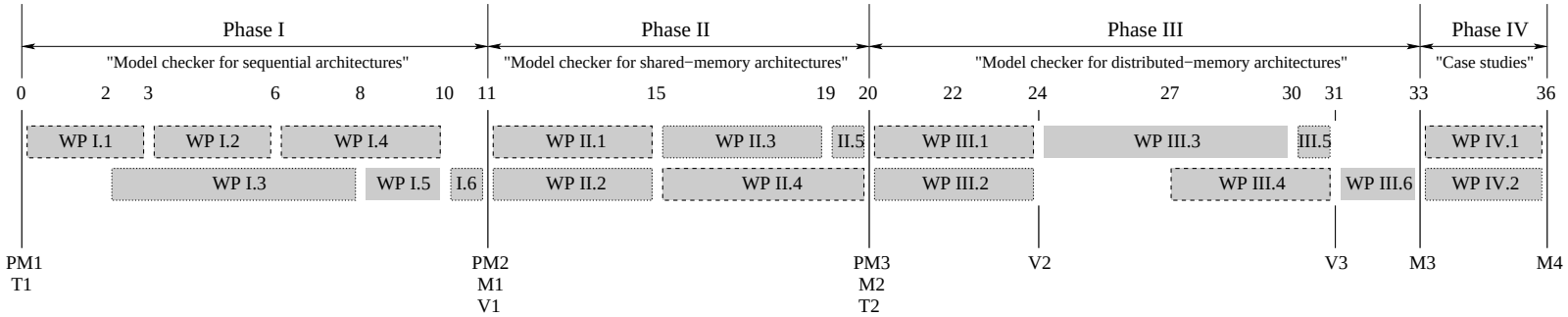
[37] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *ICCAD '93*, pp. 139–144. IEEE Comp. Soc. Press, 1993.

[38] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: Complete Reference*. MIT Press, 1996.

[39] T. Stornetta and F. Brewer. Implemention of an efficient parallel BDD package. In *DAC '96*, pp. 641–644. ACM Press, 1996.

[40] B. Yang, R.E. Bryant, D.R. O'Hallaron, A. Biere, O. Coudert, G. Janssen, R.K. Ranjan, and F. Somenzi. A performance study of BDD–based model checking. In *FMCAD '98*, vol. 1522 of *LNCS*, pp. 255–289, 1998.

Programme of Work: Diagrammatic Overview



WP I.1: Explore issues of iteration control in the context of sequential model checking
WP I.2: Devise heuristics for the partitioning of Petri net models
WP I.3: Implement model checker (heuristics for variable ordering, mechanisms for MDD−node sharing)
WP I.4: Conduct formal correctness proof of the sequential model checker within a theorem prover
WP I.5: Analyse the model checker's performance and compare it to NuSMV
WP I.6: Integrate model checker in SMART and make it web−accessible for remote execution

WP II.1: Investigate sources of parallelism and cache consistency issues for shared−memory architectures
WP II.2: Explore other algorithmic issues, e.g., regarding the minimisation of processor idle time
WP II.3: Implement the devised design(s) on a multi−processor workstation
WP II.4: Analyse the shared−memory model checker (proof of correctness & benchmarking)
WP II.5: Integrate shared−memory model checker in SMART and make it web−accessible for remote execution

WP III.1: Devise parallelisations for distributed−memory computer architectures (consider issues of scalability)
WP III.2: Achieve memory− and load−balance, investigate message−passing facilities
WP III.3: Implement the devised design(s) on PC/workstation clusters
WP III.4: Prove the distributed−memory model checker correct
WP III.5: Integrate distributed−memory model checker in SMART and make it web−accessible for remote execution
WP III.6: Devise a benchmark for asynchronous−systems verification and compare the model checker's performance

WP IV.1: Conduct case study "Analysing mode confusion"
WP IV.2: Conduct case study "Parallel algorithm verification"

LEGEND

Symbols:

WP x.y — Working package no. y in project phase x; to be led by York

WP x.y — Working package no. y in project phase x; to be led by William & Mary

WP x.y — Working package no. y in project phase x; to be led jointly

Abbreviations:

PM: Project meeting
M: Milestone
T: Training of the RA at William & Mary
V: Visit by the international collaborator of William & Mary in York

PM1−3: Initiate each project phase (except Phase IV) with a joint project meeting in order to coordinate research efforts (PM2 in York, PM1+3 at William & Mary)
V1−3: Visits by the collaborator to York for the crucial parallel algorithms' design (WP II.1−2, 3−week visit, WP III.1−3, 2−week visit) and the comparative performance analysis (WP III.6, 3−week visit)
T1: 3−week travel of the RA to William & Mary to receive training on the SMART tool
T2: 3−week travel of the RA to William & Mary to receive training on the usage of and programming on William & Mary's workstation cluster
M1−3: Each project phase concludes with a finished product: a model checker integrated into SMART and remotely executable via the www, together with an analysis of its performance and correctness
M4: A demonstration of the model checkers' utility for solving practically−relevant problems, given the large complexity of today's asynchronous systems and software