MŰEGYETEM 1782

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Extensions and Generalization of the Saturation Algorithm in Model Checking

Ph.D. Thesis Booklet

Vince Molnár

Thesis supervisor:
István Majzik, Ph.D. (BME)

Budapest
2019

1 Preliminaries

1.1 Motivations of the Research

Between 1985 and 1987, a software bug in the Therac-25 radiation therapy machine killed or severely injured at least six patients [LT93]. The problem was caused by improper synchronization between two concurrent processes, which lead to a race condition if the operator entered specific commands too fast. Race conditions are very hard to detect during testing because they appear only very rarely. They are a prime example of nondeterministic behavior, which is a mathematical abstraction of unpredictable but possible outcomes.

Incidents such as that of Therac-25 showed that software, especially concurrent software is very easy to get wrong, and the faults are very hard to detect with conventional testing approaches. By that time, formal methods in computer engineering had well-laid foundations in mathematical logic, which allowed engineers to specify, model and verify their designs in a mathematically precise way [BH14]. With a formal specification and model, the goal of formal verification is to *prove* the correctness of the modeled behavior in terms of the specification as opposed to testing certain executions. At the time of the Therac incidents, pioneering work in the automation of detecting concurrency problems has just been published by Edmund M. Clarke and Ernest A. Emerson [EC80] as well as Joseph Sifakis [QS82]: the technique of *model checking*.

A certain (abstract) behavior of a computer system can be represented by a sequence of *discrete states* that describe the system in a specific moment of time. The system would stay in a state for some amount of time, then *transition* into another one practically instantaneously. The whole behavior of the system can therefore be regarded as a graph (states as nodes and transitions as directed arcs), where multiple outgoing transitions denote nondeterministic choices (e. g. the scheduling of concurrent processes as described above). Every path, i. e. sequence of states in this graph is a possible execution of the system that might realize under some circumstances. Therefore, any potential error is also present somewhere in the graph, as an undesired state or sequence of states. The goal of model checking is to construct this *state graph* (also called *state space*) and look for the error as specified by some logic formula coming from the specification of system properties.

Obviously, the more complex the system, the more states it will have, and unfortunately the relationship is usually exponential – this is known as the *state space explosion problem*. In practice, a realistic system model would usually have so many states that it is impossible to store all of them in computer memory. One of the solutions proposed for this problem is *symbolic model checking* [Bur+92].

The main motivation of symbolic model checking is that states are usually vectors of values assumed by variables of the system, and many of these state vectors in the state space would be similar (especially in concurrent systems). For example, a thread in a program will most likely change only its local variables and some of the shared global variables, but will not affect local variables of any other thread. Symbolic model checking exploits this by characterizing *sets of state vectors* with Boolean functions (characteristic functions), which will describe the common features of and relations between the vectors in the set. For example, a function $f(x, y, z) = \neg x \vee y$ over Boolean variables x, y and z describe 6 vectors by returning *true* for exactly those triples that satisfy $\neg x \vee y$. Basic set operations needed for model checking then map to Boolean operators (union to disjunction, intersection to conjunction, complementation to negation).

As a compact representation for Boolean functions, Randal Bryant proposed binary *decision diagrams* [Bry86], which are essentially decision trees with all the identical subtrees merged. Manipulation of decision diagrams to execute logical operations can be very efficient with recursion and caching, as the merged subtrees have to be processed only once.

The recursive nature of decision diagrams has inspired a new model checking algorithm that – instead of using breadth-first search (BFS) or depth-first search (DFS) – follows the structure of the decision diagram to recursively compute the state graph through a series of submodels, each reused in the next one until the precise result is found. The algorithm suits decision diagrams very well and provides a fast and memory-efficient way to construct the decision diagram representing the set of reachable states of the system. Its goal is to expand the decision diagram that encodes the initial state node by node, in a bottom-up fashion, hence its name – *saturation* [CLS01; CMS06].

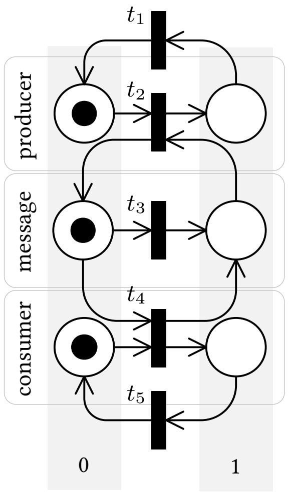
A crucial property required for the efficiency of saturation is called locality. The notion of locality means that when the system changes state in response to some event, only some of its components will actually transition to a new state, most of them will not be affected. If events are local, the exploration of the state space can be decomposed into smaller explorations on submodels with only a subset of components, considering only those events that are local on them. Saturation follows this strategy by considering a series of submodels: when *saturating* a decision diagram node, it considers only those variables whose values are yet to be considered in that subdiagram (and note those whose values led to the node) and explores the states reachable with the events local on these variables.

Saturation proved to be one of the most efficient decision diagram-based symbolic model checking algorithms, especially for concurrent systems. It was initially designed for Petri net models [CLS01], where it exploited the so-called Kroenecker condition of transitions (which allows for a very simple representation), but later improvements removed this constraint [CMS06]. A distinct variant of the algorithm is *constrained saturation* [ZC09], which can keep exploration inside a predefined set of states without modifying the transitions of the model (this problem appears e. g. when searching *backwards* from a state in an already explored state space). The problem with extending the transitions by an additional check to see if they leave the constraint set is that it destroys locality, because that check will depend on all state variables even when the original transition did not. With constrained saturation, the algorithm handles the constraint set separately and in this special case it can still exploit locality of the original transitions.

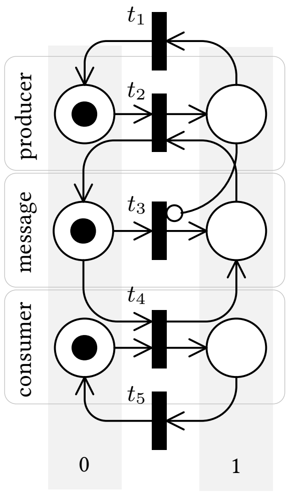
Besides the problems solved so far, there are a number of open questions related to the saturation algorithm. Even though constrained saturation solves the issue for constraint sets, it is generally hard for saturation to handle global or interdependent transitions. Such an interdependency is introduced when transitions have priorities. In this case, transitions have to consider also whether any other transition with a higher priority could be executed, which most of the time makes the transition global in the sense that it must be aware of all state variables. Prioritized transitions are common in Generalized Stochastic Petri Nets [Chi+93], which is a popular modeling formalism for stochastic systems. In this setting, symbolic model checking not only has to provide efficient state space exploration, but also has to support deriving a proper representation for numerical solvers.

Another challenge arises when saturation is used for the model checking of linear temporal logic (LTL) properties. LTL describes (infinite) executions of a system and translates to Büchi automata, an extension of finite state automata that accepts infinite words. Model checking of LTL properties involves the synchronization of the system model and the automaton translated from the negated LTL property such that the automaton reads the labels of system states as letters. If the automaton accepts an execution of the system then we have a proof that the property can be violated (as we translate the negated property). Since the automaton usually reads most of the state variables, the synchronized transitions will again lose locality. Furthermore, unlike explicit state graph-based model checkers, symbolic model checkers are usually not able to look for accepted executions before the state space is completely explored, whereas stopping upon finding a counterexample would be very much desirable.

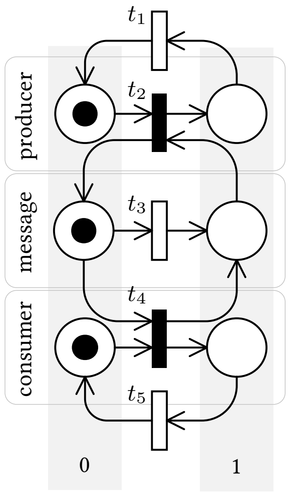
In general, saturation is weak for systems where events are not local enough. In such cases, it will degrade to BFS or DFS, which tend to produce larger intermediate decision diagrams leading to larger resource consumption and less efficient scaling. The research presented in this work



(a) Simple version.



(b) With inhibitor arc.



(c) With timed transitions.

Figure 1. Petri net models describing 3 variations of a producer-consumer system with a buffer for a single message. We treat each component as a Boolean variable which is 0 when the left place is marked and 1 when the right one.

focuses on extending saturation to handle these situations efficiently. At the end, it turns out that there is a common idea in all of these extensions that can be used to further enhance saturation on many classes of models and also generalize many different variations that were considered as different algorithms before.

1.2 Background

To understand the results of this work, the reader should have a basic understanding of how the saturation algorithm and LTL model checking works. This section will illustrate the main concepts through a running example that will be used to demonstrate the new algorithms as well.

Petri nets. Figure 1 presents three Petri net models [Mur89] of a producer-consumer system with a single message passed back and forth between them. The left-most model (1a) is a simple version where both the producer, the consumer and the message have two *places* (the circles), representing their two possible states: *ready* and *busy* in case of the producer and consumer and *unprocessed* and *processed* in case of the message. When a place is marked by a *token* (black filled circle), the component is assumed to be in the corresponding state.

States may be changed by *transitions* (rectangles). *Arcs* show how transitions consume and produce tokens from and to places (one token by default) when *fired*. Firing a transition means that a behavior is executed, changing the *marking* (the distribution of tokens on places) according to the arcs. A transition is *enabled* when it has the necessary tokens to consume and enabled transitions may be fired anytime, causing nondeterministic behavior if multiple ones are enabled.

That said, the example model has 5 possible behaviors: t_1 and t_5 will return a *busy* producer or consumer to the *ready* state, t_2 will cause a *ready* producer to remove the *processed* message and create a new, *unprocessed* one, while the producer will become *busy* with producing the next message, t_4 similarly causes the consumer to consume and process an *unprocessed* message, and t_3 represents a timeout when the *unprocessed* message simple becomes *processed*. With these transitions, we can see that exactly one place of every component will be marked in any state of the system, so we may treat each component as a Boolean variable: it is 0 when the left place is marked and 1 when the right one.

	cons.	msg.	prod.
t_1			rw
t_2		rw	rw
t_3		rw	
t_4	rw	rw	
t_5	rw		

(a) Simple version.

	cons.	msg.	prod.
t_1			rw
t_2		rw	rw
t_3		rw	**r**
t_4	rw	rw	
t_5	rw		

(b) With inhibitor arc.

	cons.	msg.	prod.
t_1	**r**	**r**	rw
t_2		rw	rw
t_3	**r**	rw	**r**
t_4	rw	rw	
t_5	rw	**r**	**r**

(c) With timed transitions.

Figure 2. Dependency matrices (DM) for the 3 variants of the example. Letters *r* and *w* stand for *read* and/or *write* dependencies, differences from the simple model are in boldface. Colors denote the highest component that is not independent from the transitions, an information used by the saturation algorithm.

The model in the middle (1b) extends the system with an *inhibitor arc* (an arc with a circle instead of an arrow). Inhibitor arcs will *disable* a transition if the connected place is marked (by default by one token). Therefore, this variant says that a timeout may only occur if the producer is *ready*, e. g. because the producer's process keeps track of timeouts and will only check when it is *ready*.

The third model (1c) uses the Generalized Stochastic Petri Net (GSPN) formalism [Chi+93], which is an extension of Petri nets with probabilistic firing and time. While simple Petri nets model only the order of transition firings, GSPNs consider a continuous time model where each transition firing happens in a precise instant of time. In this formalism, we can distinguish two types of transitions: *immediate* transitions are just like before and will fire as soon as they are enabled (possibly in a nondeterministic order), while *timed* transitions (empty rectangles) have a *firing rate* that affects how soon they will fire after becoming enabled. An important consequence of this distinction is that immediate transitions *must* fire before any enabled timed transition, adding *priorities* to the transitions.

With this in mind, the third model adds the information that the producer and consumer will actually spend time on creating and processing messages (*busy* state), and the timeout will happen after a specific amount of time, while the emitting (t_2) and consuming (t_4) of new messages will happen immediately when possible. A consequence of the implied priorities is that e. g. a timeout may not occur if the consumer is ready to consume a message.

Locality. The example model consists of 3 fundamentally asynchronous concurrent components that occasionally synchronize. Therefore, most of the transitions affect only some of the components: this is known as locality. Figure 2 show the dependency matrices for the 3 variants of the example system. Rows of the matrix are transitions, while columns are components. If a cell is empty, then the transition is independent of the component, while the letters *r* and *w* denote *read* and *write* dependencies (in Petri nets, inhibitor arcs and test arcs introduce read dependencies, while every other arc implies a read-write dependency). Additional dependencies of the second and third variant compared to the first one are in boldface.

For example, the inhibitor arc added in the second variant introduces a read-only dependency between t_3 and the producer component, while the priorities implied by immediate and timed transitions lead to read-only dependencies between timed transitions (t_1, t_3 and t_5) and every component that is read by any immediate transition (to check if any immediate transition is enabled). Notice that these additional dependencies causes the transitions to be *less local* in general.

Model checking. Model checking is an automated technique to prove that a detailed system model is indeed a model of its specification – in other words, that it is *correct* with respect to certain requirements [CGP99]. A crucial step in model checking is state space exploration. Usually, systems are not modeled with their state graph, but rather a high-level model *encoding* the state space (such as Petri nets). Model checking works on the state space, so most approaches involve some form of state space generation. An infamous problem with this approach is the *state space explosion* problem: combinatorial explosion is common when we move from a high-level model to its state space.

With regard to the requirements, linear temporal logic (LTL) is a popular formalism to describe aspects of the intended temporal behavior [Pnu77]. It can specify requirements such as fairness, e. g. all processes of a system must be scheduled in a fair way (i. e. none of them can remain idle for the rest of the time, written as). Negated LTL properties specifying *counterexamples* are often translated to Büchi automata, a variant of the finite-state automaton working on infinite sequences [Büc62]. Such an automaton is used to keep track of how the system behavior affects the satisfaction of the property: if the automaton passes some of its *accepting states* infinitely many times, then the sequence is a counterexample.

Checking LTL properties on finite-state systems is reduced to finding reachable loops in the combined state space of the system model and the Büchi automaton that include at least one *accepting* state [VW86]. In explicit, graph-based model checkers the exploration is stopped as soon as a loop is detected, leading to fast counterexample-detection. In general, the goal is to look for reachable *strongly connected components* (SCC) in the state space, because in an SCC every state is reachable from every other.

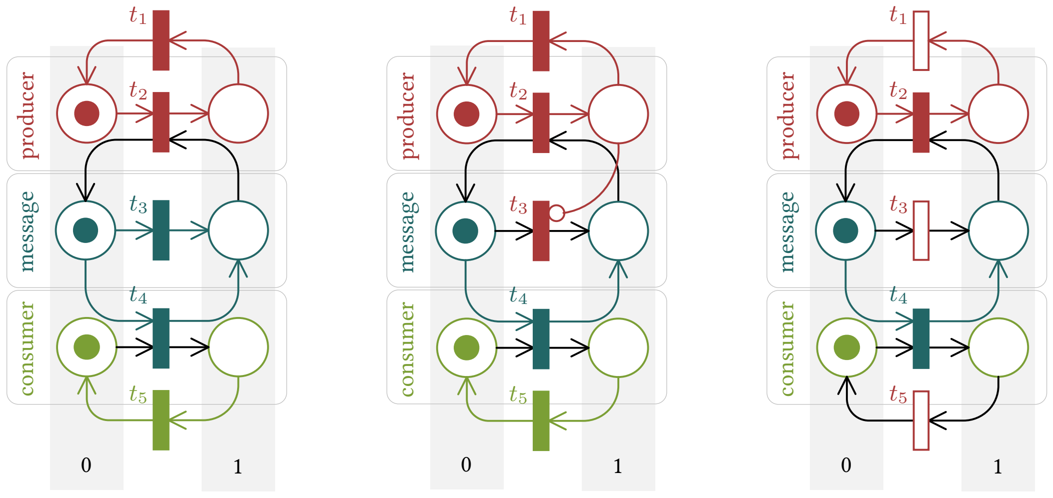
Decision diagrams. The state of a component-based system is usually represented as a vector of values, one for each component (or state variable). If transitions are local, then only some of these values will change with each firing. For example, in the simple variant, firing the timeout transition (t_3) in the initial state vector $(0, 0, 0)$ will lead to $(0, 1, 0)$, which differs only in the state of the message.

An efficient data structure for a set of vectors with common prefixes is a *decision tree*, where each node (starting from the root) is assigned to a variable, and (directed) outgoing arcs correspond to values of that variable. If we follow a path from the root through child nodes to a leaf of the tree according to the values in a vector, we can encode in the leaf whether the set represented by the decision tree contains the vector (**1**) or not (**0**). This way, common prefixes of contained vectors are not repeated in the data structure, essentially compressing the set.

If we want to exploit the common suffixes as well, we can merge the identical subtrees of the decision tree to obtain a decision diagram with exactly two "leafs": the terminal one (**1**) and zero (**0**) nodes [Bry86; MD98]. In this work, we assume that nodes on every path in the decision diagram follow the same *variable order*, and each path evaluates all variables unless they lead to **0**. Consequently, every path from the root node to the **1** node is labeled with a vector contained in the represented set following a fixed variable order.

Saturation. While decision diagrams usually offer a compact representation of large sets, computing that set – i. e. exploring the state space – can still be resource intensive. The *saturation algorithm for model checking* works directly on decision diagrams and exploits locality by recursively dividing the problem into smaller problems, caching the results of these sub-computations to avoid redundant computation [CLS01; CMS06].

Without going into every detail, the idea of saturation is simple and elegant. Saturation itself is a function that takes a decision diagram node encoding the initial states and a similarly recursive representation of the transition relation (e. g. as another decision diagram) and returns a decision diagram node encoding the set of states reachable from the initial states through a



(a) Simple version. (b) With inhibitor arc. (c) With timed transitions.

Figure 3. Submodels for saturation corresponding to each component. Colors denote the assignment of transitions to submodels. Arcs that are responsible for the assignment (introducing the dependency to the higher level) are also colored. Note that dependencies introduced by priorities are not visible (see Figure 2 for the dependency matrices).

possibly empty sequence of transition firings. This function is defined recursively based on the following observation.

Every child node of the root decision diagram node encodes a set of initial states for another model that is obtained by omitting the state variable corresponding to the root node (the *topmost* variable) and also every transition that is not independent from that variable. Since this is a *must abstraction*, everything that can happen in the abstract submodel can happen in the original model as well. Therefore, if we recursively explore the state space of this model with every set of initial states that is encoded by children of the root node, the only transitions that are not yet fired during any exploration are those that are dependent on the topmost variable. The saturate function therefore has to fire these transitions exhaustively. One last task is to check after each firing if there is any new initial state in the submodel and start a recursive exploration for those as well, which may enable even more firings for the current run of the saturate function.

The power of saturation comes from this strategy of dividing the model into smaller submodels and processing simpler subproblems enhanced with an operation cache. The efficiency of this strategy depends on how local transitions are, as well as what variable order is used. This work is concerned with the former problem, while there is ongoing research about the latter [Amp+19]. The effect of locality on the submodels is shown by the coloring of Figure 3.

1.3 Summary of Challenges

Challenge 1: Extending saturation to efficiently handle Petri nets with priorities. Generalized Stochastic Petri nets (GSPN) are a popular formalism to model stochastic systems. The efficient state space exploration of GSPNs is an important part of stochastic analysis, which can verify extrafunctional properties of a system such as performance or reliability. Can the efficiency of saturation be leveraged for prioritized Petri nets?

Challenge 2: Extending saturation to efficiently perform model checking of linear temporal logic (LTL) properties. In the model checking of LTL properties, the system behavior has to be synchronized with a Büchi automaton describing the property. To use saturation, we have to avoid the direct computation of synchronized transition relations in order to avoid losing locality. Can we modify saturation to compute the synchronous product on the fly while also exploiting locality?

Challenge 3: Finding accepting executions (that violate the property) on the fly during state space exploration with saturation. In LTL model checking, we have to look for accepted executions (i. e. counterexamples) in the synchronous product of the system model and the Büchi automaton belonging to the negated property. In a finite state space, an accepting run is always a lasso, i. e. a path leading to a cycle. Therefore, the problem can be reduced to finding reachable strongly connected components (SCC) in the state space. Explicit-state model checkers can do this on the fly, stopping immediately when an accepted execution is found. Is there a way to create an efficient on-the-fly SCC detection algorithm and incorporate it into saturation-based model checking?

Challenge 4: Generalizing the different variants of constrained saturation. Constrained saturation is an efficient take on recovering locality in special cases. Its core idea to recover locality appears in many variants solving specialized problems. Currently, these variations are considered to be separate algorithms, which hinders the opportunity to freely combine different aspects of model checking (e. g. modeling and specification formalisms). Is there a way to generalize this idea as an abstract algorithm from which constrained saturation and other variants can be instantiated?

Challenge 5: Adapting saturation to better handle models where events have global effects. Some models have mainly synchronous behavior where every transition has to be aware of most of the components or variables. In this case, saturation will not be able to exploit locality and will degrade to less efficient exploration strategies. Is it possible to adapt the ideas of saturation even in these cases? Will the resulting improvements be beneficial on concurrent, asynchronous models where transitions are local and saturation is already efficient?

The research presented in this dissertation addresses these challenges and in particular aims to remove some of the limitations of saturation to provide even more efficient algorithms. Doing so expands the class of problems where model checking is applicable in practice and facilitates the spreading of automatic formal verification tools. To this end, the dissertation presents new scientific results in the form of three new saturation-based algorithms for model checking: saturation extended for prioritized models (Thesis 1), a complete algorithm family for the incremental, on-the-fly model checking of LTL properties with saturation (Thesis 2), and finally a generalization and enhancement of the saturation algorithm itself that incorporates the discovered ideas into the original algorithm to further improve its efficiency (Thesis 3). Table 1 presents how the results relate to the described challenges.

Table 1. Relations between theses and challenges.

		Challenge				
		1	2	3	4	5
Thesis 1	*Chapter 3 of the dissertation*	•			•	
Thesis 2	*Chapters 4–5 of the dissertation*		•	•	•	
Thesis 3	*Chapter 6 of the dissertation*				•	•

2 Research Methods and New Results

Research Methods. With regard to the classification suggested by J. N. Amaral [Ama], this research follows the *build*, *formal* and *experimental* methodologies. The build methodology consists of building a novel artifact “to demonstrate that it is possible” [Ama]. Formal methodologies are used to “prove facts about algorithms and systems” [Ama], in particular the algorithms developed with the build methodology. Finally, the experimental methodology is used to evaluate a system by targeted measurements to answer specific questions.

The dissertation builds on well-established results from the fields of mathematical logic, graph theory, automata theory, as well as previously designed algorithms that are presented as background knowledge or related work. Formalisms used from the field of formal methods include Kripke structures, Petri nets, Büchi automata and linear temporal logic, and most of the algorithms introduced here are related to model checking. The presented research can be regarded as a continuation of the work of G. Ciardo on the saturation algorithm [CLS01].

The results were evaluated on an extensive set of Petri net models from the Model Checking Contest (MCC) [Kor+18], which include many industrial examples as well as artificial models to demonstrate scaling or handling of corner cases. Most of the algorithms are compared to state-of-the-art competitors, including NuSMV2 [Cim+02], nuXmv [Cav+14] and ITS-tools [Dur+11].

2.1 Analysis of Generalized Stochastic Petri Nets with Symbolic State Space Generation

As noted in Section 1.2, GSPNs introduce transition priorities into Petri net models. Timed transitions always have less priority than immediate transitions, and the formalism allows the modeler to specify priorities between immediate transitions as well. Therefore, state space exploration should be able to consider the various priority levels of the transitions, leading to the reduced transition locality as demonstrated in Figure 1c.

Decomposing transition relations. The key idea behind the result of this thesis is that fireability of a transition can be checked based on two separate aspects: *enabledness* as in Petri nets without priorities and the *highest priority* assigned to any (other) enabled transition. For prioritized Petri nets, the highest priority assigned to any enabled transition in any marking can be compiled statically before state space exploration and is independent from the representation of the transitions themselves. To encode this information, I have introduced Edge-Valued Interval Decision Diagrams (EVIDD).

The enabling region of a transition is an (infinite) box in the n dimensional space of Petri net markings, labeled with its priority, where n is the number of places in the net. For the sake of clarity, the top row of Figure 4 illustrates the concept on a simple Petri net with two places: therefore regions are rectangles or quadrants in 2-dimensional space. With one or more sets of potentially overlapping regions corresponding to every transition in the net (t_1 and t_2 in this case), the *Union-Max* operation computes a new set of disjoint regions that cover exactly the same points as the original regions (rightmost part of Figure 4), where labels are obtained as the maximum of labels on original regions that cover the new region. For regions that do not enable any transition we assign the priority level 0.

Edge-Valued Interval Decision Diagrams. An EVIDD is hybrid between of Edge-valued Decision Diagrams (EDD) [RS10] and Interval Decision Diagrams (IDD) [Tov08]: possible values of a variable are not enumerated but partitioned into intervals (like in IDDs) and each decision amounts to a portion of a value that is computed for the evaluated vector (like in EDDs, as opposed to having only a binary outcome).

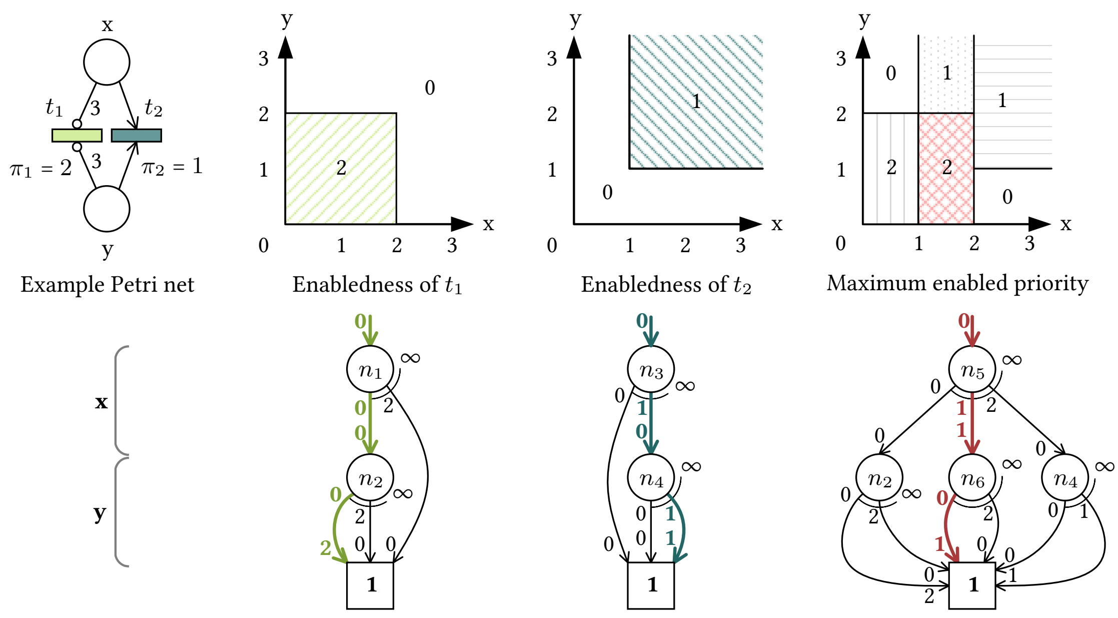


Figure 4. Regions of the space of markings defined by (transitions of) the example Petri net with priorities (top row) and corresponding EVIDD representations (bottom row). Highlighted regions are encoded by the highlighted paths in the EVIDD.

It is capable of representing the enabling (and disabling) regions of a transition with its priority as a path in the diagram: the bottom row of Figure 4 shows the EVIDD representations of the sets of regions above them. A *handle* (at the arrowhead of an arc) is a pair of a *node* and a *weight*. The weight represents a portion of the priority of the highest enabled transition belonging to the region such that the priority is the sum of weights along the path. A node represents a dimension (in this case a marking of a place) and has an ordered list of *arcs* partitioning the dimension: each arc corresponds to an interval with an inclusive lower bound defined by its label and exclusive upper bound defined by the label of the next arc of the node, or infinity (which is not associated to an arc), and points to a *child handle* for the next dimension or the terminal node. Weights are distributed automatically by reduction rules: shared weight of children of a node is pushed up to the handle of the parent node (such that there is always a child handle with weight 0). The Union-Max operation is defined recursively on handles and computes an EVIDD representing a disjoint set of regions with maximum priorities as described above. This way, a single EVIDD can encode the highest priority of enabled transitions in any marking.

With an EVIDD describing the highest priority of enabled transitions in every possible marking, fireability can be decided during the computation of the next states in saturation by traversing the EVIDD simultaneously with the decision diagram encoding the state space: if at any point the sum of integer labels exceed the priority of the current transition, it cannot fire and the resulting next state set on that level in the recursion will be empty.

Results. There is not much previous work available on the efficient processing of prioritized Petri nets with saturation. A paper by A. S. Miner et al. [Min01] describes an approach that encodes priorities in the transition relation and uses matrix-diagrams (a special kind of decision diagrams to describe relations) along with a splitting algorithm to repartition the next-state relation such that each partition is as local as possible. My approach has the advantage that the original locality is preserved in the transition relations and the modified saturation algorithm keeps track of priority-related information *for all transitions* in one place, with a compact data structure [c4]. Experimental evaluation showed that this is indeed an advantage and the EVIDD-based solution scales better than that of [Min01].

Thesis 1 I designed an algorithm to help the efficient analysis of Generalized Stochastic Petri Nets (GSPN). It extends the traditional saturation algorithm to perform a more efficient symbolic state space exploration of systems with prioritized transitions.

1.1 I introduced a new type of decision diagram called Edge-Valued Interval Decision Diagram (EVIDD) that can encode enabledness of prioritized transitions in GSPNs.

1.2 I extended the saturation algorithm to handle prioritized transitions efficiently using an EVIDD instead of encoding priorities in the transition relation.

1.3 I evaluated the algorithm and showed that it scales better than previously known approaches.

The importance of these results is that the efficient state space exploration of GSPNs can be a bottleneck of stochastic analysis, so advancements in this field will improve the whole process. Saturation-based state space exploration can handle larger state spaces, and the resulting decision diagram representations can be used to compute efficient decompositions for numerical solvers [c5]. The solvers, in turn, can process these larger state spaces with an acceptable resource budget [c7], leading to scalable stochastic analysis that is necessary to prove extra-functional requirements in safety-critical systems [j1; c6].

István Majzik was taking part in this research as my Ph.D. thesis supervisor. Implementation and experimentation were done in cooperation with Kristóf Marussy as my student, and András Vörös provided helpful comments. The results of Thesis 1 are presented in Chapter 3 of the dissertation. Related publications are the following: [j1], [c4], [c5], [c6], [c7].

2.2 Saturation-Based Incremental Model Checking for Linear Temporal Logic

Challenge 2 and 3 showed that symbolic LTL model checking has two distinct problems to solve: computing the combined state space of the system and a Büchi automaton describing the negated property, and looking for accepting cycles that represent a counterexample in the system.

Synchronizing with Büchi automata. Computing the combined state space is challenging for saturation because each transition has to be synchronized with one of the transitions of the Büchi automaton such that the transition of the automaton is labeled with an expression that is true in the target state. This implies a read-dependency between variables in the LTL property and all transitions in the system, spoiling locality and reducing the effectiveness of saturation.

The solution for this is based on a similar idea as in Thesis 1. Since the state space and the transition relation of the Büchi automaton are statically available, a data structure encoding the "enabledness" (the satisfaction of expressions on automaton transitions) can be compiled offline. If we assume that atomic propositions in the property are all comparisons between a single state variable and a constant value (and never between two variables), this structure is a finite decision diagram: the upper part consists of levels corresponding to atomic propositions, ordered by the referenced variable, where arcs are labeled by 1 (*true*) and 0 (*false*) (representing the result of evaluating the atomic proposition with a value of the referenced variable), and the lower part is a representation of a subset of the transition relation of the Büchi automaton (illustrated in Figure 5). Each subset contains the transitions that are labeled according to the evaluation of atomic propositions along the path leading to the lower part.

The saturation algorithm is extended as follows. We assume that the variable encoding the state of the Büchi automaton is on the lowest level. The algorithm will keep evaluating the atomic propositions in the previously described decision diagram until it reaches this bottom level. The transition relation of the model does not specify what happens to the automaton – instead, the reference in the terminal node is used to finish the next-state computation. This way, the information that would spoil the locality of transitions is again handled separately and in one place by

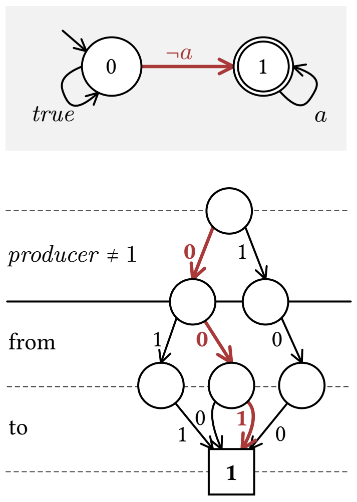


Figure 5. Nondeterministic Büchi automaton with a single accepting state (double circle) for the negation of property $GF(producer = 1)$ and its transition relation as a decision diagram.

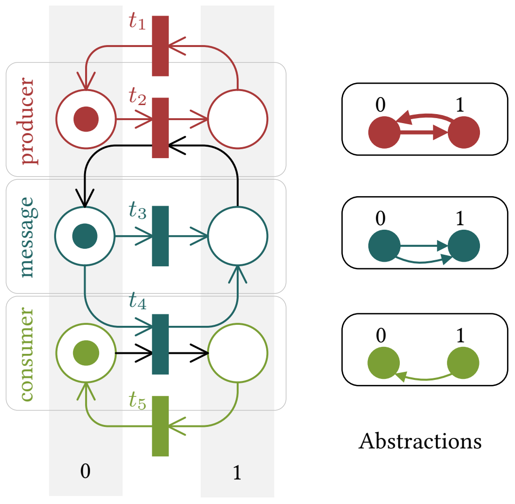


Figure 6. The simple Petri net model with abstractions for each component. Notice that there can be no cycle without the producer component – the fixed point computation algorithm will be run on that level only.

the saturation algorithm itself, allowing it to retain the recursive submodel strategy that makes it efficient.

Looking for counterexamples. The problem of finding strongly connected components with saturation has a number of known solutions (e. g. [ZC11]). These assume that the state space is already computed and apply fixed point computations to iteratively remove dead-end states until either no state is left (no SCC) or there are no more dead-ends (the remaining set contains an SCC). Although this would be enough in the LTL model checking setting, but LTL model checkers are traditionally on-the-fly algorithms, i. e. they can stop exploration as soon as a counterexample is found. This is often much faster than processing the whole state space symbolically.

The goal therefore is to apply fixed point computations more often during state space exploration. I proposed an approach that fits the recursive nature of the saturation algorithm: look for SCCs every time a node is saturated, i. e. when a submodel is fully explored, therefore dividing the problem in a similar fashion as saturation does with state space exploration. This approach has a number of advantages. First, it can detect an SCC in an abstract submodel without considering other parts of the system, which will lead to sooner counterexample detection. Secondly, the computation can be incremental in the sense that an SCC must contain at least one firing from a transition that belongs to the current level (otherwise it had been discovered on a lower level), which greatly reduces the search space.

This algorithm still has a considerable overhead and redundancy in the exploration steps. To overcome this, I have proposed two heuristics that can prove the absence of SCCs without an expensive fixed point computation.

Recurring states are states that are reached more than once during exploration. This can be either because there were more than one path leading into the state, or because it is in a cycle which has been closed there. Either way, if there are no recurring states during an exploration, there is no SCC – an observation that can be used as a cheap filter. Recurring states can be efficiently collected in the saturation algorithm and can further restrict the search space of the fixed point algorithm.

A more powerful heuristic is based on *abstractions*. Saturation itself already works on abstract submodels, but those can still be quite large, especially as more and more components are considered. By further abstraction, we can omit all the lower components as well, but this time we will assume that transitions dependent on them are *enabled* (i. e. the omitted places can have any marking – a *may abstraction* – contrary to the higher components not present in the submodel). These abstractions are shown in Figure 6 for each component. Let us constrain the state space of these abstractions to the projection of the state space encoded by the considered saturated node, which is easy to compute: every arc leading to a non-zero node encodes a state of this *node-wise abstraction* (e. g. only state “0” is considered for the consumer before “1” is discovered in a higher submodel).

It can be proved that if this abstraction does not have an SCC (which can be efficiently computed by graph algorithms), then there is no SCC in the state space encoded by the node that contains a firing of a transition that belongs to the current level. This is exactly that the fixed point-based algorithm would look for, therefore there is no point in running it. If there is an SCC, fixed point computation can be further limited to transitions whose projections constitute that SCC.

Results. These four components (computation of the combined state space, incremental SCC detection, collection of recurring states and abstractions) constitute an efficient, on-the-fly, incremental symbolic LTL model checking algorithm that outperforms most of the similar tools that were available at the time of its creation [j2; c8]. Extensive measurements on models of the Model Checking Contest showed that the algorithm is often orders of magnitude faster than its competitors, which is a significant step towards scalable LTL model checking for real-life systems.

> **Thesis 2** I designed an incremental, on-the-fly algorithm for the model checking of properties described by linear temporal logic (LTL), extending the saturation algorithm for state space generation and using it for fixed point computations.
>
> 2.1 I extended the saturation algorithm to directly generate the state space of the product system obtained by combining the system-under-analysis and a Büchi automaton describing the LTL property. The advantage of direct generation is that it avoids the explicit computation of the product transition relation.
>
> 2.2 I designed an algorithm to incrementally search for strongly connected components (SCC) in the state space during its generation by the saturation algorithm, using the saturation algorithm to compute the necessary fixed points. This approach enables on-the-fly model checking, i. e. the algorithm can terminate as soon as a witness is found.
>
> 2.3 I introduced two heuristics that complement the incremental SCC detection algorithm. Using abstraction-based techniques and the concept of recurring states, the heuristics can prove the absence of an SCC and therefore can speed up the search by preventing unnecessary fixed point computations.
>
> 2.4 I evaluated the resulting LTL model checking algorithm on models of the Model Checking Contest (MCC), comparing the runtime with three tools that represented the state of the art: the algorithm was found to be often orders of magnitude faster than its competitors.

The importance of these results is that the introduced LTL model checking algorithm often scales better than previously known approaches, while its main ideas are orthogonal to many other improvements that can be found in the literature, promising an even better result when combined. LTL is a popular formalism for specifying temporal properties, especially fairness properties, which cannot be expressed in the other popular formalism, computational-tree logic (CTL). Because of this, advancement in this direction can help in achieving wide-spread use of formal verification in (concurrent) safety-critical systems where fairness is a crucial part or precondition of properties to verify. The algorithm is implemented in the PetriDotNet framework[1] [j1; c6]

András Vörös and Tamás Bartha were taking part in this research as my B.Sc. and M.Sc. supervisors. Dániel Darvas helped to understand the saturation algorithm and wrote the code that I used as the basis of my implementation. Parts of this work are presented in my B.Sc. and M.Sc. theses [a19; a18]. The results of Thesis 2 are presented in Chapters 4–5 of the dissertation. Related publications are the following: [j1], [j2], [c6], [c8].

2.3 Enhancement and Generalization of the Saturation Algorithm

Work on the previous theses revealed common problems that appear in many different contexts: *1)* different next-state representations (including but not restricted to different types of decision diagrams) usually come with a specialized variant of the saturation algorithm and are not compatible with each other, as well as *2)* losing locality is generally a concern in every variant, whereas usually there is a workaround that retains some of the locality at least.

Conditional Locality. According to our experience in implementing saturation-based model checkers, a generic representation of transition relations that works well with saturation would be desirable to separate the algorithm from the representation. We have introduced Abstract Next-State Diagrams (ANSD) to generalize *decision diagram-like representations* by extracting their common features into an abstract interface [c4]. Even though the approaches discussed in Theses 1 and 2 could be expressed as specific implementations of this interface, incorporating the handling of the EVIDD and the transition relation of the Büchi automaton in the saturation algorithm itself had a reason: to retain locality.

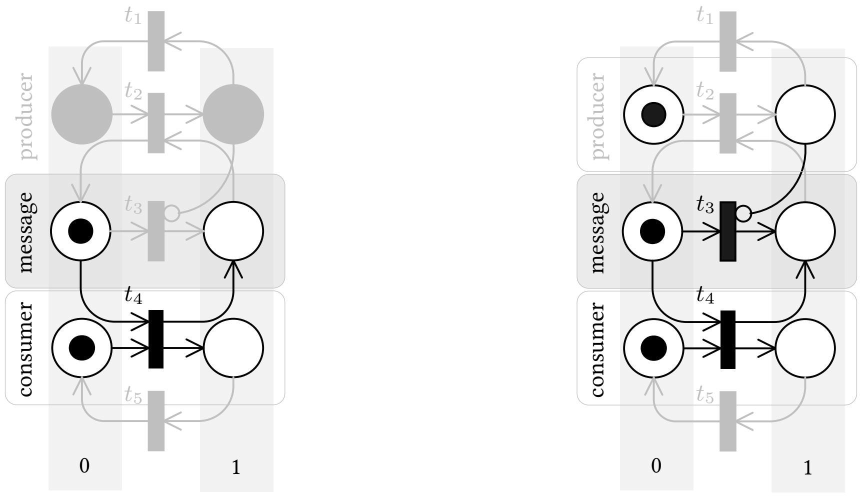
The core of this thesis is an observation that resulted in the introduction of *conditional locality*. The recursive saturation algorithm exploits the fact that transitions in an abstract submodel belonging to a decision diagram node are independent from the higher variables that are not included in the submodel, therefore their enabledness can be checked and they can be fired arbitrary many times. This is important because computing a local fixed point for the submodel requires the exhaustive firing of all transitions. But can any other transition be fired arbitrary many times?

The answer is yes: for example, transitions with read-only dependencies to higher variables could be fired arbitrary many times, because they will not change the value of the omitted state variables, *given that they are enabled*. In general, any transition that does not change the value of omitted variables can be fired arbitrary many times, even if the value of that variable affects the outcome of the firing.

Conditional locality formulates exactly this property: a transition is conditionally local *with respect to a prefix of a state vector* if firing it from a state with that prefix will result in a state with the same prefix, i. e. those values are not changed [c3]. This idea (in a simplified way) is illustrated in Figure 7: recursive abstract submodels now contain the places above the current level and also those transitions that have write dependency with the current variable and at most read-only dependency to higher levels.

The Generalized Saturation Algorithm. I have proposed a generalized version of saturation based on conditional locality that computes these submodels dynamically, automatically partitioning the transition relation to process everything on the lowest level possible [c3]. This approach enhances the saturation effect, because a larger portion of the work is performed on

[1] http://petridotnet.inf.mit.bme.hu/en/



(a) As seen by saturation. (b) As seen by the GSA.

Figure 7. The submodel of the example with inhibitor arc (Figure 1b) corresponding to the *message* component as seen by the original saturation algorithm as well as by the GSA. Notice that the GSA can decide the enabledness of t_3 but it will not change the state of the producer.

small submodels. Even better, including more transitions in the submodels lead to sub-results that are more likely to be final, yielding a faster and more memory-efficient algorithm.

The generalized saturation algorithm (GSA) works with ANSDs to represent any next-state relation. With the automatic partitioning, representations introduced in Theses 1 and 2 (EVIDDs and the decision diagram for the Büchi automaton) as well as previously published variants such as the constraint from constrained saturation [ZC09] can be dynamically intersected with the original transition relation under the ANSD abstraction layer, so the saturation algorithm itself does not have to be adapted. Furthermore, the SCC detection algorithm (along with the heuristics) of Thesis 2 can also benefit from processing transitions on a lower level, as this can speed up the computation and also result in sooner detection of SCCs (on a lower level).

Results. I have compared the GSA with the original saturation algorithm on Petri net models of the MCC to find that it has virtually no overhead when conditional locality is the same as simple locality, whereas it is often orders of magnitude faster and more memory efficient on models with read-only dependencies. The experiments imply that the GSA takes the purely beneficial ideas of saturation one step further without introducing any overhead, while also generalizing the solutions of a family of problems that were hard to solve in the context of the original algorithm. Even though this result covers some of the results in Theses 1 and 2, it is my most recent and most important contribution in this field that would not have been conceived without identifying and generalizing the common ideas in these and other preceding results.

Thesis 3 I designed an enhancement of the saturation algorithm using the concept of conditional locality. I showed that the classic constrained saturation algorithm, the extension for prioritized Petri nets (Thesis 1), as well as the extension for computing the product state space (Thesis 2) are instances of this generalized saturation algorithm.

3.1 I defined the concept of conditional locality and conditionally saturated nodes, relaxing the notion of locality used in the original saturation algorithm.
3.2 Based on conditional locality, I designed the generalized saturation algorithm (GSA) which enhances the saturation effect and therefore improves the original algorithm. I formally proved the correctness of the new algorithm.

3.3 I showed that the GSA generalizes a family of saturation variants based on constrained saturation. I described the original constrained saturation algorithm along with the extended variants proposed in this work in terms of the GSA.

3.4 I evaluated the GSA on Petri net models of the model checking contest (MCC) and showed that it may outperform the original saturation algorithm by orders of magnitudes in some cases, while it has no considerable overhead in any other case.

The importance of these results is that the saturation algorithm was already one of the most successful symbolic model checking algorithms, which has been improved with the introduction of conditional locality. One of the largest problems in model checking is state space explosion, that can be countered by better scaling and abstractions, which is combined in this result. Furthermore, this result generalizes a family of algorithms, facilitating the correctness, compatibility, a better understanding and maintainable implementation of saturation-based model checking tools by providing a common framework and general correctness proofs for the core algorithm. Finally, the ANSD abstraction layer for the transition relation and the automatic partitioning provided by the GSA enable the direct model checking of more general modeling formalisms (e. g. hierarchical state machines) which in turn facilitates the integration of model checking tools to high-level modeling tools such as the Gamma Statechart Composition Framework (developed by Bence Graics under my supervision) [c12; c13; c14].

István Majzik was taking part in this research as my Ph.D. thesis supervisor.

The results of Thesis 3 are presented in Chapter 6 of the dissertation. Related publications are the following: [c3], [c4].

3 Application of the New Results

3.1 Stochastic Analysis of Generalized Stochastic Petri Nets

Saturation for prioritized Petri nets has been used in a tool for the stochastic analysis of GSPNs, where state space exploration is performed by saturation [c4], the state space decomposition is based on decision diagrams [c5] and numerical analysis is performed by a configurable solver framework [c7]. Stochastic analysis is implemented in the PetriDotNet framework[2] [j1; c6] as well, which has been used in the dependability analysis of automotive embedded systems.

3.2 LTL Model Checking

The saturation-based on-the-fly, incremental LTL model checking algorithm is also implemented in the PetriDotNet framework that is freely available online. With this, PetriDotNet offers a comprehensive set of analysis algorithms built into a graphical editor, allowing users to model, simulate and analyze Petri nets and stochastic Petri nets using properties expressed either as invariants, CTL or LTL formulas or extra-functional metrics. The tool has been used in a number of independent projects as discussed in [j1]. The LTL model checking algorithm described in the thesis was the first one to successfully verify LTL properties on the so-called PRISE model, which models a safety procedure in the Paks Nuclear Power Plant in Hungary detecting primary-to-secondary leakage accidents [Ném+09].

3.3 The Generalized Saturation Algorithm

Even though the GSA is fairly new, we are working on exploiting its properties in the Theta Configurable Abstraction Refinement-based Verification Framework [Tót+17] developed by our research group. The framework offers abstraction-based software model checking capabilities

[2]http://petridotnet.inf.mit.bme.hu/en/

with counterexample-guided abstraction refinement (CEGAR), as well as various input languages to facilitate integration with modeling tools. A promising research direction is the combination of saturation and CEGAR. Concurrently, we are working on integrating the framework with the Gamma Statechart Composition Framework [c12; c13; c14], where the verification of component-based systems will benefit from the power of saturation algorithm. The Gamma framework is used in industrial projects for the modeling and analysis of embedded systems as well as for code and test generation.

3.4 Use Cases for Model Checking

A part of my research not strictly related to the theses was about the use cases of model checking, which guided my work towards practical applicability. In the CECRIS project[3], I have developed a model checking-based methodology for automated Failure Mode and Effects Analysis (FMEA) of software [j10; c16]. The approach relies on injecting non-deterministically activating faults into the program, then analyzing its behavior with a model checker for different purposes. For FMEA, one is concerned about whether the fault activation will result in a system-level failure, which can be specified separately. The approach can also be used to evaluate fault tolerance mechanisms and error detectors by checking the conformance of the fault-free version and the faulty version coupled with the evaluated mechanism. The results of the project including my work has been published in a book [b17].

In cooperation with my student Levente Bajczi we have also introduced and investigated a novel problem domain involving parallel programs executed on multi-core processors, where the memory controller has a crucial role on the correctness of the system as a whole [c11]. With the widespread use of custom-built processors in embedded systems, faulty memory controllers are less and less rare, party because they tend to be very complex, and also because hardware verification techniques are now able to reveal the problem even after release. Fortunately, these purpose-built chips often run a single program during their lifetime, and the activation of the fault is often linked to specific circumstances. If our program will never encounter those circumstances, then there should be no activation and the problem will remain dormant. Our work showed that today's model checkers are not very well suited for this problem, opening new research questions and challenges with the formal definition of a new use case.

[3]FP7–Marie Curie (IAPP) number 324334

4 Publication List

Number of publications:	17
Number of peer-reviewed journal papers (written in English):	4
Number of articles in journals indexed by WoS or Scopus:	4
Number of publications (in English) with at least 50% contribution of the author:	7
Number of peer-reviewed publications:	17
Number of independent citations:	18

4.1 Publications Linked to the Theses

	Journal papers	International conference and workshop papers
Thesis 1	[j1]*	[c4]*, [c5], [c6]*, [c7]
Thesis 2	[j1]*, [j2]	[c6]*, [c8]
Thesis 3	–	[c3] [c4]*

* Papers marked by an asterisk belong to more than one theses.
This classification follows the faculty's Ph.D. publication score system.

Journal Papers

[j1] András Vörös, Dániel Darvas, Ákos Hajdu, Attila Klenik, Kristóf Marussy, Vince Molnár, Tamás Bartha, and István Majzik. Industrial applications of the PetriDotNet modelling and analysis tool. *Science of Compututer Programming* 157, 2018, pp. 17–40. DOI: 10.1016/j.scico.2017.09.003. URL: https://doi.org/10.1016/j.scico.2017.09.003.
▷ *The implementation of the PetriDotNet framework is a joint work of the authors, based on the original work of Bertalan Szilvási. The saturation-based LTL model checking algorithm is my own contribution, supervised by A. Vörös and T. Bartha.*

[j2] Vince Molnár, András Vörös, Dániel Darvas, Tamás Bartha, and István Majzik. Component-wise incremental LTL model checking. *Formal Aspects of Computing* 28(3), 2016, pp. 345–379. DOI: 10.1007/s00165-015-0347-x. URL: https://doi.org/10.1007/s00165-015-0347-x.
▷ *Own contribution, joint paper with B.Sc., M.Sc. and Ph.D. supervisors.*

International Conference and Workshop Papers

[c3] Vince Molnár and István Majzik. Saturation enhanced with conditional locality: Application to Petri nets. In: Susanna Donatelli and Stefan Haar (eds.), *Application and Theory of Petri Nets and Concurrency - 40th International Conference, PETRI NETS 2019, Aachen, Germany, June 23-28, 2019, Proceedings*, Lecture Notes in Computer Science, vol. 11522, pp. 342–361. Springer, 2019. DOI: 10.1007/978-3-030-21571-2_19. URL: https://doi.org/10.1007/978-3-030-21571-2%5C_19.
▷ *Own contribution, joint paper with Ph.D. supervisor.*

[c4] Kristóf Marussy, Vince Molnár, András Vörös, and István Majzik. Getting the priorities right: Saturation for prioritised Petri nets. In: Wil M. P. van der Aalst and Eike Best (eds.), *Application and Theory of Petri Nets and Concurrency - 38th International Conference, PETRI NETS 2017, Zaragoza, Spain, June 25-30, 2017, Proceedings*, Lecture Notes in Computer Science, vol. 10258, pp. 223–242. Springer, 2017. DOI: 10.1007/978-3-319-57861-3_14. URL: https://doi.org/10.1007/978-3-319-57861-3%5C_14.
▷ *Own contribution, joint paper with M.Sc. and Ph.D. supervisors and my student K. Marussy.*

[c5] Kristóf Marussy, Attila Klenik, Vince Molnár, András Vörös, István Majzik, and Miklós Telek. Efficient decomposition algorithm for stationary analysis of complex stochastic Petri net models. In: Fabrice Kordon and Daniel Moldt (eds.), *Application and Theory of Petri Nets and Concurrency - 37th International Conference, PETRI NETS 2016, Toruń, Poland, June 19-24, 2016. Proceedings*, Lecture Notes in Computer Science, vol. 9698, pp. 281–300. Springer, 2016. DOI: 10.1007/978-3-319-39086-4_17. URL: https://doi.org/10.1007/978-3-319-39086-4%5C_17.
▷ *The stochastic analysis algorithm is the contribution of K. Marussy. The macro-state decomposition algorithm is my own contribution.*

[c6] András Vörös, Dániel Darvas, Vince Molnár, Attila Klenik, Ákos Hajdu, Attila Jámbor, Tamás Bartha, and István Majzik. PetriDotNet 1.5: Extensible Petri net editor and analyser for education and research. In: Fabrice Kordon and Daniel Moldt (eds.), *Application and Theory of Petri Nets and Concurrency - 37th International Conference, PETRI NETS 2016, Toruń, Poland, June 19-24, 2016. Proceedings*, Lecture Notes in Computer Science, vol. 9698, pp. 123–132. Springer, 2016. DOI: 10.1007/978-3-319-39086-4_9. URL: https://doi.org/10.1007/978-3-319-39086-4%5C_9.
▷ *The implementation of the PetriDotNet framework is a joint work of the authors, based on the original work of Bertalan Szilvási. The saturation-based LTL model checking algorithm is my own contribution, supervised by A. Vörös and T. Bartha.*

[c7] Kristóf Marussy, Attila Klenik, Vince Molnár, András Vörös, Miklós Telek, and István Majzik. Configurable numerical analysis for stochastic systems. In: *2016 International Workshop on Symbolic and Numerical Methods for Reachability Analysis (SNR)*, pp. 1–10. Apr. 2016. DOI: 10.1109/SNR.2016.7479383.
▷ *Contribution of my students K. Marussy and A. Klenik, supervised by A. Vörös, M. Telek, I. Majzik and myself.*

[c8] Vince Molnár, Dániel Darvas, András Vörös, and Tamás Bartha. Saturation-based incremental LTL model checking with inductive proofs. In: Christel Baier and Cesare Tinelli (eds.), *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, Lecture Notes in Computer Science, vol. 9035, pp. 643–657. Springer, 2015. DOI: 10.1007/978-3-662-46681-0_58. URL: https://doi.org/10.1007/978-3-662-46681-0%5C_58.
▷ *Own contribution, joint paper with B.Sc., M.Sc. and Ph.D. supervisors.*

Local Events

[l9] Vince Molnár and András Vörös. Synchronous product automaton generation for controller optimization. In: *ASCONIKK 2014: Extended Abstracts. I. Information Technologies for Logistic Systems*, pp. 22–29. Veszprém, Hungary: University of Pannonia, Dec. 2014.
▷ *The tableau automaton-based product computation algorithm is the contribution of A. Vörös and shares ideas with this work.*

4.2 Additional Publications (Related to Use Cases and Applications of the Theses)

Journal Papers

[j10] Vince Molnár and István Majzik. Model checking-based software-FMEA: Assessment of fault tolerance and error detection mechanisms. *Periodica Polytechnica Electrical Engineering and Computer Science* 61(2), 2017, pp. 132–150. DOI: https://doi.org/10.3311/PPee.9755. URL: https://pp.bme.hu/eecs/article/view/9755.

International Conference and Workshop Papers

[c11] Levente Bajczi, András Vörös, and Vince Molnár. Will my program break on this faulty processor? - Formal analysis of hardware fault activations in concurrent embedded software. *Transactions on Embedded Computing Systems* 18(5s), 2019, pp. 1–21. DOI: https://doi.org/10.1145/3358238.

[c12] Vince Molnár, Bence Graics, András Vörös, István Majzik, and Dániel Varró. The Gamma statechart composition framework: Design, verification and code generation for component-based reactive systems. In: Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (eds.), *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pp. 113–116. ACM, 2018. DOI: 10.1145/3183440.3183489. URL: https://doi.org/10.1145/3183440.3183489.

[c13] Bence Graics and Vince Molnár. Mix-and-match composition in the Gamma framework. In: Béla Pataki (ed.), *Proceedings of the 25th PhD Mini-Symposium*, pp. 24–27. Budapest, Hungary, 2018.

[c14] Bence Graics and Vince Molnár. Formal compositional semantics for Yakindu statecharts. In: Béla Pataki (ed.), *Proceedings of the 24th PhD Mini-Symposium*, pp. 22–25. Budapest, Hungary, 2017.

[c15] Vince Molnár and István Majzik. Constraint programming with multi-valued decision diagrams: A saturation approach. In: Béla Pataki (ed.), *Proceedings of the 24th PhD Mini-Symposium*, pp. 54–57. Budapest, Hungary, 2017.

[c16] Vince Molnár and István Majzik. Evaluation of fault tolerance mechanisms with model checking. In: Béla Pataki (ed.), *Proceedings of the 23rd PhD Mini-Symposium*, pp. 30–33. Budapest, Hungary, 2016.

Book Chapters

[b17] Valentina Bonfiglio, Francesco Brancati, Francesco Rossi, Andrea Bondavalli, Leonardo Montecchi, András Pataricza, Imre Kocsis, and Vince Molnár. Composable framework support for software-FMEA through model execution. In: Bondavalli Andrea and Brancati Francesco (eds.), *Certifications of Critical Systems – The CECRIS Experience*, pp. 183–200. Delft, Netherlands: River Publishers, 2017.

4.3 Additional Work

[a18] Vince Molnár. Advanced Saturation-based Model Checking. Master's Thesis. Budapest University of Technology and Economics, 2014. URL: https://diplomaterv.vik.bme.hu/en/Theses/Szaturacio-alapu-modellellenorzes.

[a19] Vince Molnár. Szaturáció alapú modellellenőrzés lineáris idejű tulajdonságokhoz [in Hungarian; Saturation-based model checking for linear temporal properties]. Bachelor's Thesis. Budapest University of Technology and Economics, 2013. URL: http://petridotnet.inf.mit.bme.hu/publications/BSThesis2013_Molnar.pdf.

[a20] Vince Molnár and Dániel Segesdi. Múlt és jövő: Új algoritmusok lineáris temporális tulajdonságok szaturáció-alapú modellellenőrzésére [in Hungarian; Future and past: New algorithms for the saturation-based model checking of linear temporal properties]. Scientific Students' Association Report. 1st prize, national 2nd prize (OTDK 2015). 2013. URL: http://petridotnet.inf.mit.bme.hu/publications/TDK2013_MolnarSegesdi.pdf.
▷ *The Past-LTL to Büchi automaton translation and the automaton simplification algorithm*

is the contribution of D. Segesdi. The automata-theoretic model checking algorithm based on saturation is my own contribution.

[a21] Vince Molnár. Szaturáció alapú modellellenőrzés lineáris idejű tulajdonságokhoz [in Hungarian; Saturation-based model checking for linear temporal properties]. Scientific Students' Association Report. 2nd prize. 2012. URL: http://petridotnet.inf.mit.bme.hu/publications/TDK2012_Molnar.pdf.

References

[Ama] José Nelson Amaral. About Computing Science Research Methodology. URL: https://webdocs.cs.ualberta.ca/~c603/readings/research-methods.pdf.

[Amp+19] Elvio Gilberto Amparore, Gianfranco Ciardo, Susanna Donatelli, and Andrew S. Miner. i_{Rank}: A variable order metric for DEDS subject to linear invariants. In: Tomás Vojnar and Lijun Zhang (eds.), *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II*, Lecture Notes in Computer Science, vol. 11428, pp. 285–302. Springer, 2019. DOI: 10.1007/978-3-030-17465-1_16.

[BH14] Dines Bjørner and Klaus Havelund. 40 years of formal methods - some obstacles and some possibilities? In: Cliff B. Jones, Pekka Pihlajasaari, and Jun Sun (eds.), *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*, Lecture Notes in Computer Science, vol. 8442, pp. 42–61. Springer, 2014. DOI: 10.1007/978-3-319-06410-9_4.

[Bry86] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* C-35(8), 1986, pp. 677–691. DOI: 10.1109/TC.1986.1676819.

[Büc62] J. Richard Büchi. On a decision method in restricted second order arithmetic. In: Ernest Nagel, Patrick Suppes, and Alfred Tarski (eds.), *Proc. of the 1960 International Congress on Logic, Methodology and Philosophy of Science*, pp. 1–11. Stanford Univ. Press, 1962.

[Bur+92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation* 98(2), 1992, pp. 142–170. DOI: 10.1016/0890-5401(92)90017-A. (Visited on 09/08/2014).

[Cav+14] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Alberto Griggio, Marco Roveri, and Stefano Tonetta. *The nuXmv Symbolic Model Checker.* Tech. rep. Fondazione Bruno Kessler, 2014.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[Chi+93] Giovanni Chiola, Marsan Marco Ajmone, Gianfranco Balbo, and Gianni Conte. Generalized stochastic Petri nets: A definition at the net level and its implications. *IEEE Trans. Software Eng.* 19(2), 1993, pp. 89–107.

[Cim+02] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV 2: an opensource tool for symbolic model checking. In: Ed Brinksma and Kim G. Larsen (eds.), *Computer Aided Verification*, Lecture Notes in Computer Science, vol. 2404, pp. 359–364. Springer, 2002. URL: http://dx.doi.org/10.1007/3-540-45657-0_29.

[CLS01] Gianfranco Ciardo, Gerald Lüttgen, and Radu Siminiceanu. Saturation: an efficient iteration strategy for symbolic state-space generation. In: *Proc. of the 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 328–342. 2001.

[CMS06] Gianfranco Ciardo, Robert Marmorstein, and Radu Siminiceanu. The saturation algorithm for symbolic state-space exploration. *International Journal on Software Tools for Technology Transfer* 8(1), 2006, pp. 4–25. DOI: 10.1007/s10009-005-0188-7. (Visited on 08/24/2014).

[Dur+11] Alexandre Duret-Lutz, Kaïs Klai, Denis Poitrenaud, and Yann Thierry-Mieg. Self-loop aggregation product – A new hybrid approach to on-the-fly LTL model checking. In: Tevfik Bultan and Pao-Ann Hsiung (eds.), *Automated Technology for Verification and Analysis*, Lecture Notes in Computer Science, vol. 6996, pp. 336–350. Springer, 2011. DOI: 10.1007/978-3-642-24372-1_24.

[EC80] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In: J. W. de Bakker and Jan van Leeuwen (eds.), *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, Lecture Notes in Computer Science, vol. 85, pp. 169–181. Springer, 1980. DOI: 10.1007/3-540-10003-2_69.

[Kor+18] F. Kordon et al. Complete Results for the 2018 Edition of the Model Checking Contest. http://mcc.lip6.fr/2018/results.php. 2018. (Visited on 2018).

[LT93] N. G. Leveson and C. S. Turner. An investigation of the Therac-25 accidents. *Computer* 26(7), 1993, pp. 18–41.

[Mar+16b] Kristóf Marussy, Attila Klenik, Vince Molnár, András Vörös, Miklós Telek, and István Majzik. Configurable numerical analysis for stochastic systems. In: *2016 International Workshop on Symbolic and Numerical Methods for Reachability Analysis (SNR)*, pp. 1–10. 2016. DOI: 10.1109/SNR.2016.7479383.

[MD98] D. Michael Miller and Rolf Drechsler. Implementing a multiple-valued decision diagram package. In: *Proc. of the 28th IEEE International Symposium on Multiple-Valued Logic*, pp. 52–57. 1998. DOI: 10.1109/ISMVL.1998.679287.

[Min01] Andrew S Miner. Efficient solution of GSPNs using canonical matrix diagrams. In: *Petri Nets and Performance Models, 2001. Proceedings. 9th International Workshop on*, pp. 101–110. 2001.

[Mur89] Tadao Murata. Petri nets: properties, analysis and applications. *Proceedings of the IEEE* 77(4), 1989, pp. 541–580. DOI: 10.1109/5.24143.

[Ném+09] Erzsébet Németh, Tamás Bartha, Csaba Fazekas, and Katalin M. Hangos. Verification of a primary-to-secondary leaking safety procedure in a nuclear power plant using coloured Petri nets. *Reliability Engineering & System Safety* 94(5), 2009, pp. 942–953. DOI: 10.1016/j.ress.2008.10.012.

[Pnu77] Amir Pnueli. The temporal logic of programs. In: *Proc. of the 18th Annual Symposium on Foundations of Computer Science*, pp. 46–57. IEEE Computer Society, 1977. DOI: 10.1109/SFCS.1977.32.

[QS82] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In: Mariangiola Dezani-Ciancaglini and Ugo Montanari (eds.), *International Symposium on Programming*, pp. 337–351. Springer Berlin Heidelberg, 1982.

[RS10] Pierre Roux and Radu Siminiceanu. Model checking with edge-valued decision diagrams. In: *Proc. of the 2nd NASA Formal Methods Symposium*, pp. 222–226. 2010.

[Tót+17] Tamás Tóth, Ákos Hajdu, András Vörös, Zoltán Micskei, and István Majzik. Theta: A framework for abstraction refinement-based model checking. In: Daryl Stewart and Georg Weissenbacher (eds.), *2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017*, pp. 176–179. IEEE, 2017. DOI: 10.23919/FMCAD.2017.8102257.

[Tov08] Alexey A. Tovchigrechko. Efficient symbolic analysis of bounded Petri nets using interval decision diagrams. PhD thesis. Brandenburg University of Technology, Cottbus-Senftenberg, Germany, 2008.

[VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In: *Proc. of the Symposium on Logic in Computer Science*, pp. 332–344. IEEE Computer Society, 1986.

[ZC09] Yang Zhao and Gianfranco Ciardo. Symbolic CTL model checking of asynchronous systems using constrained saturation. In: *Proc. of the 7th Int. Conf. Automated Technology for Verification and Analysis*, pp. 368–381. 2009.

[ZC11] Yang Zhao and Gianfranco Ciardo. Symbolic computation of strongly connected components and fair cycles using saturation. en. *Innovations in Systems and Software Engineering* 7(2), 2011, pp. 141–150. DOI: 10.1007/s11334-011-0146-3. (Visited on 08/24/2014).