Polynomial Formal Verification of Sequential Circuits

Caroline Dominik

Institute of Computer Science
University of Bremen/DFKI
28359 Bremen, Germany
cardom@uni-bremen.de

Rolf Drechsler

Institute of Computer Science
University of Bremen/DFKI
28359 Bremen, Germany
drechsler@uni-bremen.de

***Abstract*—Recently, the concept of *Polynomial Formal Verification* (PFV) has been introduced and successfully applied to several classes of functions, allowing complete verification under resource constraints. But so far, all studies were carried out for combinational circuits only.**

In this paper we show how the concept of PFV can be extended to *sequential circuits*. As a first case study we show for counters that PFV can be performed, even though they have an exponential number of states, i.e., they can be fully formally verified within polynomial upper bounds on run-time and memory requirement.

***Index Terms*—circuit design, correctness, verification, test, formal methods, BDD**

I. Introduction

With the invention of the transistor back in 1947, the cornerstone for the digital revolution was laid. As a fundamental building block, the transistor enabled the development of digital circuits. Their mass production revolutionized the field of electronics, finally leading to computers, embedded systems, as well as the internet. Hence, the impact of digital hardware on society, as well as economy, was and is tremendous.

Over the last decades, the enormous growth of complexity of integrated circuits continued as expected. As modern electronic devices are getting more and more ubiquitous, the fundamental issue of functional correctness becomes more important than ever. This is evidenced by many publicly known examples of electronic failures with disastrous consequences. Within the last decades, this included e.g. the Intel Pentium bug in 1994 (costs: $475 million), New York blackout in 2003 (estimated costs: $1.1 billion), or a design flaw in Intel's Sandy bridge chipset in 2011 (costs: $1 billion).

Such costly mistakes can only be prevented by applying rigorous verification to the circuits before they get to production. A lot of effort has been put into developing efficient verification techniques by both academic and industrial research. Only recently, the industry has recognized the great importance of automated formal verification (see e.g. functional safety standards such as ISO 26262). Without interaction of a user being necessary, the formal methods can be applied at a larger scale. Hence, in the last few years, this research area has become increasingly active. Essentially, the goal of automated formal verification is to automatically prove that an implementation is correct with respect to its specification. Depending on what an implementation is and what a specification comprises, different verification problems arise.

Unfortunately, most of the verification problems encountered in the domain of electronic circuits are computationally intractable. Consider for example the two following core problems: *Combinational Equivalence Checking* (CEC) and *Symbolic Reachability Analysis* (SRA).

1) CEC involves checking the equivalence of two combinational circuits (i.e. without memory elements), where one circuit acts as the specification and the other (often an optimized version) is the implementation [1]. CEC is coNP-complete via reduction to tautology checking using a miter circuit.
2) SRA computes the set of reachable states from a set of initial states and is fundamental to temporal logic model checking, which checks whether a Boolean or word-level sequential circuit implementation satisfies a temporal logic specification. In SRA, sequential behavior is viewed as a transaction system and symbolically encoded by logic formulas providing a very succinct representation. SRA is computationally even harder than CEC: the Boolean case is PSPACE-complete and word-level SRA is EXPSPACE-complete [2].

In practice, the most efficient techniques reduce formal verification to a single, or a series of, decision problem(s) in propositional logic. There are two main approaches of decision procedures for propositional logic. The first is based on converting the problem into a functionally canonical form. The most prominent example is a *Binary Decision Diagram* (BDD) [3]. The second is an intelligent systematic search for a satisfying assignment employed by modern satisfiability (SAT) solvers [4]. It is obviously unavoidable that the decision procedures also have very high complexity. SAT was the first problem shown to be NP-complete [5]. The construction of BDDs is #P-complete via reduction to propositional model counting (#SAT) and problems in the #P-complete class are widely believed to be harder than NP-complete.

Thus, all these approaches suffer from the problem, that a guarantee for the feasibility of a verification within the given resources cannot be given. Or, stated differently, the required

Parts of this work have been supported by DFG within the Reinhart Koselleck Project *PolyVer: Polynomial Verification of Electronic Circuits* (DR287/36-1).

resources are unknown. This also includes, that no prediction about the performance of the algorithms is possible.

The concept of *Polynomial Formal Verification* (PFV) has been recently presented in the context of combinational circuits (see [6], [7]) and will be reviewed in the next section. The core idea is to provide polynomial upper bounds for the space and time complexity of the verification algorithms. This can be ensured by varying the proof engine and focusing on specific classes of functions. While studies for combinational circuits have been provided, this paper is the first to focus on handling sequential circuits. We show, that model checking can be modified, so that it is possible to fully verify some sequential circuits efficiently, even if the underlying *Finite State Machine* (FSM) has an exponential sequential depth. In a case study, counters, as well as *Serial In Serial Out* (SISO) shift registers are studied and it is shown that they can be polynomially verified.

The paper is structured as follows: At first, the idea of PFV is explained in more detail in Section II. There, prior achievements for combinational circuits are reviewed and the difficulties of applying PFV to sequential circuits are discussed. The case study is covered in Section III, including an experimental analysis, and in Section IV the results are summarized and future directions are given.

II. Polynomial Formal Verification

Since the introduction of formal proof techniques in the context of circuit verification, it has been observed that they might become expensive in terms of run-time and space requirement. For instance, in [3] it was already observed that BDDs are not well suited for representing multiplier functions.

While various results on sizes of the final representation have been derived (see [8] for an overview), to ensure efficiency, ***the complete verification task has to be considered*** as well. A trivial example is, that the BDD of the output of a miter circuit for a correct CEC is only the constant 0 node, while the memory for intermediate computations might be very large. In general, this observation is not new (see e.g. [9]), but in the following, it is researched how these intermediate memory peaks can be avoided.

A. Combinational Circuits

So far, there have been several successful examples for PFV of combinational circuits. In [10], polynomial time and space complexities for the verification of floating point adders are achieved. The explosive growth of BDDs during the symbolic simulation of the circuit is prevented by adding a problem-specific case splitting. Furthermore, [11] shows PFV for a very simple RISC-V processor proposed in [12]. With a divide-and-conquer approach, each instruction is viewed separately, so that all functional units can be verified with polynomial resources. In [13], PFV is applied to an ALU of a RISC-V processor with an extended instruction set. With a hybrid proof engine, the suitability of BDD-level methods for adder circuits and the suitability of word-level methods for multipliers can be combined to efficiently verify the entire ALU. While the so far mentioned proofs for PFV have all been conducted manually, [14] discusses the automatic creation of human-readable proofs for PFV. As an example, automatic proofs by induction are created for polynomial upper bounds for the size of a given BDD.

B. Sequential Circuits

The methods for sequential verification have not yet been thoroughly analyzed with respect to PFV. The already mentioned approach of [11] addresses the sequential behavior of a multi-cycle RISC-V processor by simulating the circuit over several clock cycles. This technique allows a combinational approach to the sequential behavior, but limits PFV to circuits with small sequential depth.

In the sequential domain, different scenarios can be considered:

1) Analogously to CEC, with *Sequential Equivalence Checking* (SEC), the behavior of two sequential circuits can be compared. If SEC is carried out based on a miter circuit, the equality has to hold for any sequence of inputs, which increases the computational complexity compared to CEC. This miter circuit can be modeled by an FSM and analyzed with SRA, to ensure that no state exists, which is only reachable by one of the two circuits. Therefore, SEC is a special case of model checking.
2) The more general case of model checking is to analyze the FSM of only one circuit, to make sure all required temporal properties hold. As already mentioned, deciding if a property holds is most efficiently done based on either BDDs or SAT.
    a) By symbolically representing the model and sets of states with BDDs, it is possible to significantly increase the state space of a system, to which model checking can be applied.
    b) On the other hand, bounded model checking (see Chapter 10 of [15]) unrolls the FSM for some k steps and in that way transforms the sequential problem to a combinational problem, which can be expressed by a SAT-formula. This is possible if the sequential depth of the underlying FSM is tractable, e.g. if the sequential behavior is due to a pipeline with a small number of stages.

However, PFV must fail for these scenarios with circuits that have an exponential sequential depth. It is easy to see that e.g. for an n-bit counter, which counts up to 2^n, based on these methods, an exponential run-time cannot be avoided. Other approaches have to be considered, like covering the exponential state space by an induction proof, which can be extended by generating additional invariants, or reducing the state space with partial order reduction (see Chapter 6 of [15]). However, these methods require a separate view of each circuit.

In this paper, we therefore focus on modifying BDD-based symbolic model checking, so that the underlying algorithm remains polynomial even if the circuit has an exponential sequential depth.

III. Case Study: Counters & SISO Shift registers

We start with an overview of the used concepts, to make this paper self-contained.

A. Notation and definition

1) Binary Decision Diagrams: A BDD [3] is a directed, acyclic graph, that represents a Boolean function $f : B^n \to B$. Each internal node represents a variable x_i with $0 \leq i < n$ and carries out the Shannon decomposition $f = \overline{x}_i f|_{\overline{x}_i} + x_i f|_{x_i}$. Here, $f|_{x_i}$ denotes the restriction of f by assigning $x_i = 1$, whereas $f|_{\overline{x}_i}$ denotes the restriction by $x_i = 0$. Each path describes an assignment of variables, the terminal nodes with values 0 or 1 give the value, which f evaluates to for the given assignment (see e.g. Fig. 1). The BDD is ordered if all variables follow a given order $\{x_0, x_1, \ldots, x_{n-1}\}$. The BDD is reduced if all isomorphic subgraphs and redundant nodes are removed. For this paper, we assume that all BDDs are reduced and ordered. The size $|G|$ of a BDD G is given by the number of internal nodes. BDDs can be manipulated with several operators. With $Apply(op, G, H)$ two BDDs G and H are combined by a binary operation op. In this paper, the operations conjunction "$G \wedge H$" and disjunction "$G \vee H$" are used. The number of necessary steps and the size of the result are both in $\mathcal{O}(|G| \cdot |H|)$. Computing the restriction $G|_{x_i=l}$ has a run-time in $\mathcal{O}(|G|)$. Using both of these operators, existential abstraction $\exists x_i f := f|_{\overline{x}_i} \vee f|_{x_i}$ can be implemented.

2) BDD-based Symbolic Model Checking: Model checking (see Chapter 8 of [15] for more details) is a formal method to verify the temporal behavior of a system. This behavior is given as a formal model, e.g. an FSM, which can be defined by a set of states S, a set of initial states $I \in S$ and a transition relation $T \subseteq S \times S$. T and each set of states are symbolically represented with BDDs. The model is verified using SRA for the initial states I. The algorithm starts with a set of reachable states $S_r = \emptyset$ and a frontier set of states reachable within one step $F = I$. Recursively, S_r is updated by $S_r := S_r \vee F$ and the new frontier $F := image(F, T) \wedge \overline{S}_r$ is computed, until $F = \emptyset$ signals that no new states are reachable. Each image computation $image(F, T) := rename(\exists S(F \wedge T))$ gives the set of successors of F. Here, the set $S = \{s_0, \ldots, s_{n-1}\}$ denotes the current state variables, whereas $S' = \{s'_0, \ldots, s'_{n-1}\}$ denotes the successor state variables. $\exists S$ stands for the recursive computation of $\exists s_i(\exists S)$ for all s_i in S (in the order given by the variable order) until S is the empty set. $rename$ exchanges each s'_i with the according s_i. This algorithm is limited by a possible state space explosion.

3) Circuits: We analyze several simple sequential, synchronous circuits which are defined as follows: A full n-bit counter "FC_n" consists of n flip-flops which initially are all set to 0. With each clock signal, the counter counts up by 1 in binary. Once it reaches the value $2^n - 1$, the next clock cycle resets all flip-flops to 0. An n-bit modulo-m counter "$M_m C_n$" analogously counts up to $m - 1$ for some $0 < m \leq 2^n$. It resets to 0 after reaching $m - 1$, to start counting again. In the same manner, it resets to 0 for all remaining values v with $m \leq v < 2^n$. An n-bit SISO shift register "$SISO_n$" consists of n flip-flops $ff_0, \ldots, ff_{n-1}$ and a data input d_0. With each clock signal, the value of each ff_i is shifted to ff_{i+1} and ff_0 takes the value of d_0. These circuits are given as FSMs, a state s is encoded by $s_i := ff_i$ where s_i is the i-th variable of s. Input bits can be seen as labels of the transitions, in the BDD of T each input bit is treated as an additional current state variable, so that for k input bits and n flip-flops the current state is a tuple $(d, s) := d_0, \ldots d_{k-1}, ff_0, \ldots ff_{n-1}$. The transitions then are described by a function $f : B^{k+n} \to B^n$.

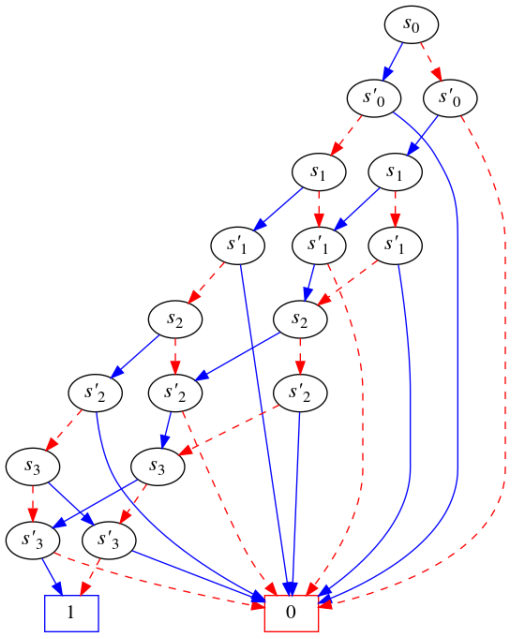

Fig. 1: BDD for the transition relation of a FC_4.

B. Theoretical analysis: Size of the transition relation

The size of the BDD T is central to the resource demands of SRA. Therefore, we first show that $|T|$ has a linear upper bound for the chosen circuits with a pair-wise variable order.

Lemma III.1. *Let BDD T describe the transition relation of a full n-bit counter with the pair-wise variable order $\{s_0, s'_0, s_1, s'_1, \ldots, s_{n-1}, s'_{n-1}\}$. Then $|T| \leq 5 \cdot n$ holds.*

Proof. Consider the variables of a state s from the *Least Significant Bit* (LSB) s_0 to the *Most Significant Bit* (MSB) s_{n-1}. During counting, they can be divided into two phases:

1) The counting affects the variable (the value is flipped).
2) The counting does not affect the variable anymore (the value is kept).

Therefore, the values of each pair of variables (s_i, s'_i) strongly depend on each other and further depend on the phase of the previous variables. T describes flipping the value during Phase 1 with $s_i \overline{s}'_i + \overline{s}_i s'_i$. Note, that $\overline{s}_i s'_i$ describes the transition from Phase 1 to Phase 2. T describes keeping the value during Phase 2 with $s_i s'_i + \overline{s}_i \overline{s}'_i$. Hence, T has at most five nodes for each pair and overall at most $5 \cdot n$ nodes. □

Example III.1. *The BDD for a full 4-bit counter can be seen in Fig. 1. The leftmost path describes Phase 1, the remaining*

nodes describe Phase 2 of the proof of Lemma III.1. It can be observed, that each value is flipped during Phase 1. Phase 1 continues for a flip from 1 *to* 0*. However, flipping from* 0 *to* 1 *transitions into Phase 2, where each value is kept.*

Lemma III.2. *Let BDD T describe the transition relation of an n-bit modulo-m counter with the variables ordered as reversed pairs $\{s_{n-1}, s'_{n-1}, s_n, s'_n, \ldots, s_0, s'_0\}$. Then $|T| \leq 10 \cdot n$ holds.*

Proof. Here, each s_i is viewed from MSB s_{n-1} to LSB s_0. As in the proof of Lemma III.1, the variables of a pair (s_i, s'_i) depend on each other and on the previous phase. In this case, however, the counter resets for several states. Therefore, T additionally differentiates between counting up and resetting. The phases for counting are:

1) The counting does not affect the variable yet (the value is kept).
2) The counting affects the variable (the value is flipped).

Phase 2 is again described by $s_i\overline{s}'_i + \overline{s}_i s'_i$ and Phase 1 by $s_i s'_i + \overline{s}_i \overline{s}'_i$ with $\overline{s}_i s'_i$ being the transition from Phase 1 to Phase 2. Analogously, this needs at most five nodes. For the reset, any value is flipped to 0. This is described as $(s_i + \overline{s}_i)\overline{s}'_i$ with one node. Further, T separates counting and resetting by storing the maximum value $m-1$. During Phase 1 of counting, an additional path describes all values v with a similar beginning to $m-1$ but $v \neq m-1$. For each s_i, exclusively only the value is kept, which s_i has in $m-1$. This is either $s_i s'_i$ or $\overline{s}_i \overline{s}'_i$, which needs two nodes. These nodes cannot be combined with the nodes for Phase 1, because this beginning cannot be followed by any value during counting. Analogously, two additional nodes describe the reset of only this value to 0. Hence, at most ten nodes are used per s_i and overall at most $10 \cdot n$ nodes are used in T. Note, that reversing the pair-wise ordering separates counting and resetting more clearly. Otherwise, T needs an extra node to decide if a flip of 1 to 0 belongs to counting or resetting. □

Example III.2. *The maximum value of a* 4*-bit modulo-*15 *counter in binary is* 1110*. As the counter counts up, keeping the value $s_2 = 1$ during Phase 1 of the proof of Lemma III.2 can have two meanings: If $s_3 = 0$ is kept as well, the counter is not yet close to the maximum value. However, if $s_3 = 1$ is kept, $s_1 = 1$ cannot be kept as well, because then* 1111 *would be reachable. Therefore, T has two separate paths for those meanings.*

Lemma III.3. *Let BDD T describe the transition relation of an n-bit SISO shift register with the pair-wise variable order $\{d_0, s'_0, s_0, s'_1, s_1, s'_2, \ldots, s_{n-2}, s'_{n-1}, s_{n-1}\}$. Then $|T| = 3 \cdot n$.*

Proof. The only dependencies are $s'_0 = d_0$ and $s'_{i+1} = s_i$ for $0 \leq i < n-1$. This gives $d_0 s'_0 + \overline{d}_0 \overline{s}'_0$ for the pair (d_0, s'_0) and $s_i s'_{i+1} + \overline{s}_i \overline{s}'_{i+1}$ for each pair (s_i, s'_{i+1}). T always needs three nodes for this. The value of s_{n-1} is shifted out of the circuit and needs no nodes. Hence, T consists of $3 \cdot n$ nodes. □

Example III.3. *The BDD for a* 3*-bit SISO shift register can be seen in Fig. 2.*

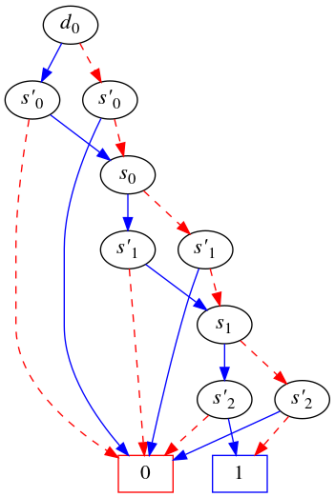

Fig. 2: BDD for the transition relation of a $SISO_3$.

C. Theoretical analysis: Restricted Domain Model Checking

We now analyze the resource demands of SRA for the considered circuits. Typically, SRA has a single initial state and exhaustively calculates a frontier set until it is empty. To obtain polynomial upper bounds even for circuits with an exponential sequential depth, we introduce a new variant of model checking, which we call ***Restricted Domain Model Checking*** (RDMC) in the following. For RDMC, we use several sets of initial states I and only compute the image $image(I, T)$ once for each set. For a circuit with k input bits, n flip-flops and transitions described by $f : B^{k+n} \to B^n$, each of those sets I is obtained by restricting the domain of f $dom(f)$ by assigning one variable first to 1 and then to 0. For each input variable d_i with $0 \leq i < k$, this gives the two initial state sets $dom(f)|_{d_i}$ and $dom(f)|_{\overline{d}_i}$. The sets for each state variable s_j with $0 \leq j < n$ are accordingly $dom(f)|_{s_j}$ and $dom(f)|_{\overline{s}_j}$. The set of all initial state sets is called $\mathbb{I}$. That way, the set based image computation is executed $2 \cdot (k+n)$ times. The reached states of a single initial state $a \in B^{k+n}$ are given by $\bigcap_{\{I | I \in \mathbb{I} \wedge a \in I\}} image(I, T)$.

Example III.4. *RDMC of a full* 3*-bit counter is described in Table I. In the first column, the state bits s_0, s_1 and s_2 are each first restricted by* 0 *and then by* 1*. The second column shows the sets of initial states I given by these restrictions. The results per image computation are in the third column. To verify, for example, that only state* 3 *can be reached by state* 2*, the initial sets $dom(f)|_{\overline{s}_2}$, $dom(f)|_{s_1}$ and $dom(f)|_{\overline{s}_0}$ are relevant, because they contain state* 2*. The according results are shown in bold, their intersection is $\{1,2,3,4\} \cap \{3,4,7,0\} \cap \{1,3,5,7\} = \{3\}$.*

Remark III.1. *RDMC is possible, if the final intersection gives exactly one reachable state per initial state $a \in B^{k+n}$.*

Example III.5. *Table I shows RDMC of a* 3*-bit modulo-*5 *counter and a* 2*-bit shift register. For both, the final results of all single initial states are single values. It also shows a*

TABLE I: EXAMPLES OF RDMC WITH 3 BITS.

d_j or s_i	$I \in \mathbb{I}$	$image(I,T)$			
		FC_3	M_5C_3	M_6C_3	$SISO_2$
$\overline{d}_0$ or $\overline{s}_2$	{0,1,**2**,3}	**{1,2,3,4}**	{1,2,3,4}	{1,2,3,4}	{0,1}
d_0 or s_2	{4,5,6,7}	{5,6,7,0}	{0}	**{5,0}**	{2,3}
$\overline{s}_1$	{0,1,4,5}	{1,2,5,6}	{1,2,0}	**{1,2,5,0}**	{0,2}
s_1	{2,3,6,7}	**{3,4,7,0}**	{3,4,0}	{3,4,0}	{1,3}
$\overline{s}_0$	{0,2,4,6}	**{1,3,5,7}**	{1,3,0}	**{1,3,5,0}**	{0,1,2,3}
s_0	{1,3,5,7}	{2,4,6,0}	{2,4,0}	{2,4,0}	{0,1,2,3}

3-bit modulo-6 counter, where the final intersection for 4 gives $\{5,0\}$ (the according result sets are in bold).

It is necessary to analyze for which circuits RDMC is applicable. It is apparent, that RDMC is possible for circuits with a bijective transition relation such as any full n-bit counter. However, the class of circuits can be extended, because this model checking variant has distinct results for any n-bit SISO shift register and for n-bit modulo-m counters with $m = 2^n - 2^c + 1$ for $0 \leq c \leq n$ as well.

Definition III.1. *A **restriction set** R is a set of assignments $\{x_0 = l_0, x_1 = l_1, \dots\}$ with $l_0, l_1 \in \{0,1\}$. Then $f|_R$ and $B|_R$ denote a Boolean function f and a set of Boolean variables B restricted by assignments $x_0 = l_0, x_1 = l_1, \dots$.*

Definition III.2. *A function $f : B^s \to B^n$ is **injective on restricted subsets** if a partition of $dom(f)$ by restriction sets $R_0, R_1, \dots$ exists, so that for each $b \in range(f)$ there is a $B^s|_{R_i}$ with $\forall a \in B^s : f(a) = b \iff a \in B^s|_{R_i}$. That way, each $B^s|_{R_i}$ contains all values of $dom(f)$ with a similar image and the function mapping each $B^s|_{R_i}$ to this image is injective.*

Example III.6. *The function for the state transitions of a 3-bit modulo-5 counter can be partitioned by restriction sets $\{\overline{s}_2\overline{s}_1\overline{s}_0\}, \{\overline{s}_2\overline{s}_1 s_0\}, \{\overline{s}_2 s_1\overline{s}_0\}, \{\overline{s}_2 s_1 s_0\}, \{s_2\}$. Such a partitioning is only possible, because the modulo value is according to $m = 2^n - 2^c + 1$ with $c = 2$. No restriction sets can be found for the 3-bit modulo-6 counter of Example III.5.*

Theorem III.1. *Let function $f : B^{k+n} \to B^n$ describe the state transitions of a circuit with n flip-flops and k input bits. RDMC can ensure the correct behavior of this circuit, if f is injective on restricted subsets.*

Proof. To ensure the correctness of the circuit, the equation

$$\forall a \in B^{k+n} : \bigcap_{\{I | I \in \mathbb{I} \wedge a \in I\}} image(I,T) = \{image(a,T)\} \quad (1)$$

must hold. The equality "=" holds if both "$\supseteq$" and "$\subseteq$" hold:
"$\supseteq$": Because $\forall a \in B^{k+n} : \bigcap_{\{I | I \in \mathbb{I} \wedge a \in I\}} I = \{a\}$ holds.
"$\subseteq$": This would not hold, if there is a $b \in B^n$ with $f(a) \neq b$ and each $I \in \mathbb{I}$ with $a \in I$ contains at least one value $c \in B^{k+n}$ with $f(c) = b$. However, as f is injective on restricted subsets, restriction sets R_0 for $f(a)$ and R_1 for b with $R_0 \neq R_1$ must exist. Therefore, there has to be at least one variable d_i (or s_j), that R_0 and R_1 assign differently, which implies that the initial state sets $dom(f)|_{d_i}$ or $dom(f)|_{\overline{d}_i}$ contain a and no value c with the image b. □

We now analyze the resource demands of RDMC using the results about $|T|$ of each viewed circuit.

Theorem III.2. *The time and space demands of RDMC of a full n-bit counter with variables ordered as pairs are in $\mathcal{O}(n^2)$ and $\mathcal{O}(n)$, respectively.*

Proof. The circuit has n flip-flops and a transition relation T. For SRA, $image(I,T) := rename(\exists S(I \wedge T))$ is executed $2 \cdot n$ times with $I \in \mathbb{I}$. Each execution consists of three operations:

1) $I \wedge T$: As stated in Section III-A1, the run-time has to be in $\mathcal{O}(|T| \cdot |I|)$. I always consists of exactly one node, therefore, applying the conjunction takes $\mathcal{O}(|T|)$ steps.
2) $\exists S$ for current state variables S: Recursively compute $\exists s_i(\exists S\ T)$ for all s_i in S. As said in Section III-A1, $\exists s_i f := f|_{\overline{s}_i} \vee f|_{s_i}$ and the run-time for restriction is linear in the number of nodes in the BDD. Since the result of Step 1 is used, the number of steps is again in $\mathcal{O}(|T|)$. The number of steps for the disjunction is accordingly in $\mathcal{O}(|T|^2)$. This holds for each recursion, because the size of the result of each recursion is in $\mathcal{O}(|T|)$. Applying $\exists s_i T$ can only increase the BDD size, if s_i divides two groups of variables, that are functionally independent from each other without s_i and the variable order does not separate the variables of both groups. Because the variables in S are removed along the variable order, each existential quantification is applied to a subgraph of T. Here, no variable in T can separate the functional dependencies like this.
3) Rename each s'_i to s_i: Simultaneous substitution can be done by traversing the results of Step 2 once and building the resulting BDD according to each node. This is in $\mathcal{O}(|T|)$ because the final result of Step 2 is used.

Hence, the overall number of steps is in $\mathcal{O}(|T|^2)$.
The size of all BDDs created during SRA is in $\mathcal{O}(|T|)$. Per execution, T and at most three more BDDs during Step 2 are used in parallel. Only the final result is kept, of which there are at most $2 \cdot n$. The space demands are therefore in $\mathcal{O}(|T|)$. Using Lemma III.1, III.2 and III.3, the overall time and space demands are in $\mathcal{O}(n^2)$ and $\mathcal{O}(n)$. □

Theorem III.3. *The time and space demands of RDMC of an n-bit modulo-m counter with $m = 2^n - 2^c + 1$ for $0 \leq c \leq n$ and a variable order as reversed pairs are in $\mathcal{O}(n^2)$ and $\mathcal{O}(n)$.*

Proof. This is analogous to the proof of Theorem III.2. □

Theorem III.4. *The time and space demands of RDMC of an n-bit SISO shift register with variables ordered as pairs are in $\mathcal{O}(n^2)$ and $\mathcal{O}(n)$.*

Proof. This is analogous to the proof of Theorem III.2. □

These results regarding SRA of the considered circuits enable the use of SEC with polynomial resources. Comparing the results of two circuits for each of the $2 \cdot (n + k)$ executions remains polynomial, because BDDs are canonical. Analogously, property checking or a comparison to a given specification of results for the SRA can be done polynomially, as long as each specification is given with respect to the $2 \cdot (n + k)$ initial sets instead of each single state.

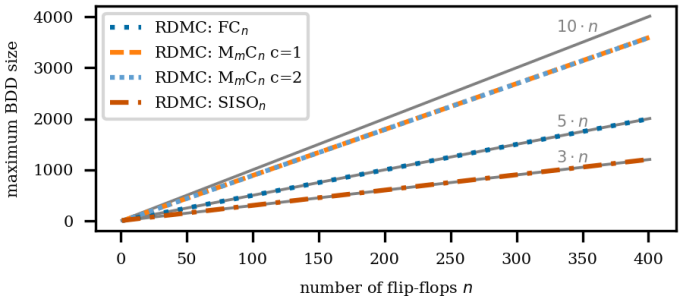

Fig. 3: Maximum BDD size during RDMC for different n.

D. Experimental study

To verify the theoretical results, RDMC has been implemented in C++. CUDD [16] was used for BDD operations. The setup has a 3.6 GHz AMD Ryzen 5 CPU and 16 GB RAM. SRA was computed for the considered circuits for different number of flip-flops n with $0 \leq n \leq 400$. For modulo-m counters the values $m = 2^n - 2^c + 1$ and $c \in \{1, 2\}$ were used. The maximum size of BDDs computed during RDMC is shown in Fig. 3. The upper limits for the size of each transition relation T given by Lemma III.1, III.2 and III.3 are marked by grey lines. As expected, no BDDs bigger than this upper limit are created. The limit is less accurate for n-bit modulo-m counters, because the maximum number of nodes is not necessary for all variables. Fig. 4 shows the run-time of RDMC. To compare the run-time to $\mathcal{O}(|T|^2)$, for each type of circuit the function $a \cdot |T|^2$ was added as a solid, grey line, with $|T|$ being the according upper limit. A small factor $a << 1$ adjusts the scale of $|T|^2$ to the scale of seconds. The measured run-times are close to these grey lines, which demonstrates that the time complexity is polynomial and not exponential with respect to $|T|$. Fig. 4 further shows the run-time of standard symbolic model checking for the same testcases (marked with "SMC"). This grows exponentially with respect to $|T|$ for counters and exceeds 10 minutes for $n = 27$ with full counters and $n = 25$ with modulo counters. Additional BDD optimizations could not significantly improve these run-times, because the number of needed steps would still grow exponentially. While the run-time for the SISO shift registers is significantly shorter compared to RDMC, this testcase still showed how RDMC can be applied to circuits with inputs.

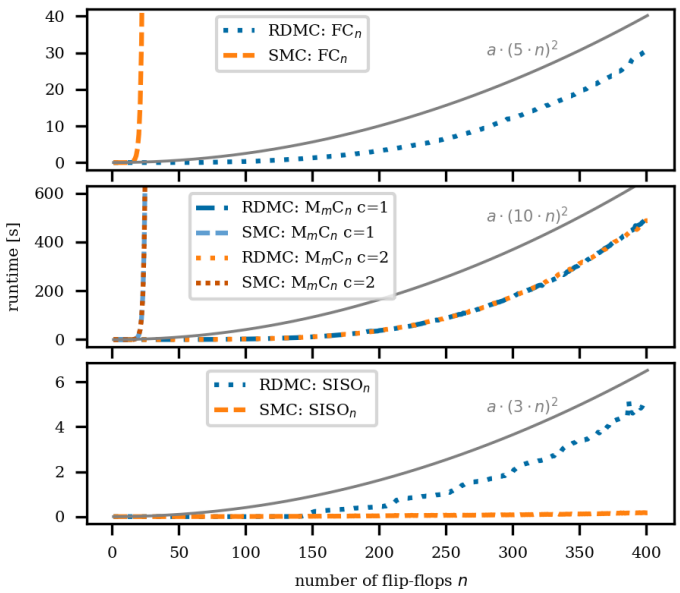

Fig. 4: Run-time during model checking for different n.

IV. Conclusion and Future Directions

This paper takes a first step to extend PFV towards circuits with sequential behavior. The current achievements in the area of PFV for combinational circuits were reviewed and the challenges of sequential verification were discussed. With RDMC, we introduced a variant of model checking, with which a polynomial run-time, in spite of exponential sequential depth, is possible. This was applied to counters, as well as to SISO shift registers.

After this first application of PFV to sequential circuits, deeper research in that area is necessary. This includes analyzing more circuits, as well as other methods. As already mentioned, sequential circuits with exponential depth could alternatively be addressed with partial order reduction or inductive proofs. The automatic generation of proofs for PFV of sequential behavior is of interest as well, as suggested in [14] for the combinational case.

References

[1] D. Brand, "Verification of large synthesized designs," in *ICCAD*, 1993, pp. 534–537.

[2] G. Kovásznai, H. Veith, A. Fröhlich, and A. Biere, "On the complexity of symbolic verification and decision problems in bit-vector logic," in *Mathematical Foundations of Computer Science*, 2014, pp. 481–492.

[3] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Comp.*, vol. 35, no. 8, pp. 677–691, 1986.

[4] A. Biere, M. Heule, H. V. Maaren, and T. Walsh, *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[5] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing*, 1971, p. 151–158.

[6] R. Drechsler, "PolyAdd: Polynomial formal verification of adder circuits," in *DDECS*, 2021, pp. 99–104.

[7] R. Drechsler and A. Mahzoon, "Polynomial formal verification: Ensuring correctness under resource constraints," in *ICCAD*, 2022, pp. 1–9.

[8] I. Wegener, *Branching Programs and Binary Decision Diagrams - Theory and Applications*. SIAM Monographs on Discrete Mathematics and Applications, 2000.

[9] J. Jain, A. Narayan, A. Sangiovanni-Vincentelli, C. Coelho, R. K. Brayton, S. P. Khatri, and M. Fujita, "Decomposition techniques for efficient ROBDD construction," in *FMCAD*, 1996, pp. 419–434.

[10] J. Kleinekathöfer, A. Mahzoon, and R. Drechsler, "Polynomial formal verification of floating point adders," in *DATE*, 2023, pp. 1–2.

[11] L. Weingarten, K. Datta, A. Kole, and R. Drechsler, "Complete and efficient verification for a RISC-V processor using formal verification," in *DATE*, 2024.

[12] S. Ahmadi-Pour, V. Herdt, and R. Drechsler, "The MicroRV32 framework: An accessible and configurable open source RISC-V cross-level platform for education and research," *Journal of Systems Architecture - Embedded Software Design (JSA)*, vol. 133, pp. 1–12, 2022.

[13] R. Drechsler and A. Mahzoon, "Divide and verify: Using a divide-and-conquer strategy for polynomial formal verification of complex circuits," in *DATE*, 2023, pp. 1–2.

[14] R. Drechsler and M. Schnieber, "Next-generation automatic human-readable proofs enabling polynomial formal verification," in *MEMOCODE*, 2023, pp. 122–125.

[15] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, *Handbook of Model Checking*. Springer Cham, 2018.

[16] F. Somenzi, "CUDD: CU decision diagram package release 2.5.1," 2015. [Online]. Available: https://github.com/ivmai/cudd