Using MaxBMC for Pareto-Optimal Circuit Initialization

Sven Reimer Matthias Sauer Tobias Schubert Bernd Becker

Institute for Computer Science
Albert-Ludwigs-Universität Freiburg
79110 Freiburg, Germany
{ reimer | sauerm | schubert | becker }@informatik.uni-freiburg.de

***Abstract*—In this paper we present *MaxBMC*, a novel formalism for solving optimization problems in sequential systems. Our approach combines techniques from symbolic SAT-based Bounded Model Checking (BMC) and incremental MaxSAT, leading to the first MaxBMC solver.**

In traditional BMC safety and liveness properties are validated. We extend this formalism: in case the required property is satisfied, an optimization problem is defined to maximize the quality of the reached witnesses. Further, we compare its qualities in different depths of the system, leading to Pareto-optimal solutions.

We state a sound and complete algorithm that not only tackles the optimization problem but moreover verifies whether a global optimum has been identified by using a complete BMC solver as back-end.

As a first reference application we present the problem of circuit initialization. Additionally, we give pointers to other tasks which can be covered by our formalism quite naturally and further demonstrate the efficiency and effectiveness of our approach.

I. Introduction

In recent years, Bounded Model Checking (BMC) has become a more and more popular technique in the area of formal verification. Unlike traditional Model Checking, in which an entire system gets validated, BMC considers a certain number of temporal steps, starting from a given set of initial states: The implementation is bounded to a given length k, validating whether the property under consideration (i. e., the specification) is satisfied for length k or a counterexample is computed. The length k is incremented until either the system has been unrolled up to a user-defined bound or a witness (i. e., trace) for reaching the negated property under consideration has been found.

Today's symbolic BMC is predominantly based on SAT solvers as first introduced in [1]. Concerning SAT-based BMC, many accelerating techniques like incremental SAT solving [2] have been developed to speed up the search process and hence the practicability. In that particular case, the basic idea is to reuse already learned information from former SAT solver calls in following similar SAT instances. Likewise, there are several other methods dedicated to the BMC framework [3], [4].

A drawback of the bounded concept in BMC is its incompleteness: Without modifications classical BMC is not able to prove the absence of a witness. Hence, methods have been developed in order to prove the unreachability of the property under consideration. Examples for such approaches are k-induction [5] and Craig interpolation [6].

BMC has a range of applications [7], [8], [9], that in particular includes checking liveness and safety properties. Safety properties tackle the question, whether it is possible to reach a *bad state*, whereas liveness properties deal with the question whether a *good state* can always be reached in the system (starting from a set of initial states in both scenarios). However, classical BMC does not allow to specify the quality of traces in the context of a general application. Instead, any trace that requires the least number of unrollings may be returned.

This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS) and the Research Training Group "Embedded Microsystems" (GRK 1103).

We tackle this question by applying an optimization problem, leading to a novel application-oriented perception of maximization problems in sequential circuits, which we call *MaxBMC*. Going beyond classical properties, MaxBMC incorporates a user-given symbolically defined metric that labels each witness with a *quality* value, representing its grade in the context of a broader general application. In MaxBMC we are not only interested in the reachability (or unreachability) of a target property, but moreover we want to determine whether the quality of a solution can be improved further. Either within the same depth, or by applying additional time steps of the sequential system. Therefore, we do not stop unrolling the sequential system once we reached the good/bad state, as better witnesses may require a higher depth. This results in *Pareto-optimal* qualities wrt. the sequence length.

Depending on the application, the qualities of the witnesses do not necessarily in-/decrease monotonically with the number of time steps to the maximal possible solution. Instead, they may vary within some bounds that depend on the problem formulation and the depth of the trace. Hence, we prove whether the found upper and lower bounds represent local optima that can be improved further in order to localize *solution bounds*. To do so, we apply techniques for early fix-point computation from BMC. In particular, in our implementation we use Craig interpolants [6] to identify these bounds as early as possible.

In this paper we formalize MaxBMC and state a sound and complete algorithm using symbolic representations. In particular, we present the first *incremental* MaxSAT solver and combine it with techniques known from *symbolic BMC* to solve the encoded representation of the combination of implementation, specification, and optimization efficiently. Additionally, we demonstrate the applicability of MaxBMC by solving the problem of *initializing* (or synchronizing) a given sequential circuit [10], [11]. Solving this problem requires to find a maximal set of flip-flops that can be functionally set to a known value, starting from a completely unknown circuit state. Furthermore, we give pointers to additional applications.

The reminder of the paper is structured as follows. First, we give a brief overview of SAT, the related MaxSAT problem, and BMC in Section II. Our formalism of MaxBMC as well as an algorithm to handle this formalism is highlighted in Section III, while the implementation details of our reference solver are presented in Section IV. Finally, we introduce some applications in Section V and discuss experimental results in Section VI. Section VII concludes the paper.

II. Preliminaries

In this section we briefly introduce the satisfiability problem (SAT), the related optimization problem MaxSAT, and Bounded Model Checking (BMC). The interested reader is referred to [12] for further details.

A. Satisfiability and MaxSAT

Given a propositional formula φ, SAT seeks an assignment A of the Boolean variables $\mathcal{V}$ occurring in φ such that φ evaluates to logic 1. An assignment A is a function $A : \mathcal{V} \to \{0, 1\}$. We call A a *model* of φ iff $\varphi(A) = 1$ holds. In that case we also call φ satisfiable. Otherwise, if there is no A for which $\varphi(A) = 1$ holds, the formula is said to be unsatisfiable. Propositional formulae are typically given in conjunctive normal form (CNF), which is a conjunction of clauses. A clause is a disjunction of literals, and

a literal is a variable $v \in \mathcal{V}$ or its negation $\neg v$. Therefore φ is satisfied iff all clauses are satisfied, and a clause is satisfied iff at least one literal of the clause is assigned to 1.

Today's state-of-the-art SAT solvers [13], [14] are based on the DPLL algorithm [15], which *decides* assignments to the variables and *deduces* resulting assignments (also referred to as implications). A key feature is the concept of conflict driven clause learning (CDCL): in case a clause is unsatisfied due to the assignments (also called *conflict*), the solver creates a reason in form of a clause for this conflict. We say such a clause is *derived* from φ. The solver resolves the conflict by withdrawing the conflicting assignments and by adding the derived clause to φ, which prevents the solver from choosing the same conflicting assignment again.

Many SAT-related formalisms have been introduced in recent decades. One prominent example is the *Maximum Satisfiability* problem (MaxSAT). Intuitively, in a MaxSAT problem we try to satisfy *as many clauses as possible* in φ. In this context the clauses are also called *soft clauses*. There are several natural extensions of MaxSAT like *Weighted MaxSAT* and *Partial MaxSAT*. In the former extension the clauses are labeled with non-negative weights and the goal is to maximize the sum of the weights of the satisfied clauses. In the latter extension there are additional so-called *hard clauses*, which *must* be satisfied, whereas the soft clauses are treated as in MaxSAT. Likewise SAT, one obtains a model which indicates the MaxSAT objective: the number of soft clauses (or the sum of the clause weights) which are satisfied simultaneously. In the following we always refer to MaxSAT within the meaning of MaxSAT or one of its extensions.

Modern MaxSAT solvers use different techniques for handling the optimization constraints. In general, one can distinguish between three main approaches: *branch-and-bound* [16], *core-guided* [17] and *iterative* [18] algorithms. Common iterative methods for example encode the maximization property of the soft clauses as cardinality constraints and add them to the original formula via adder-, counter-, or sorter-networks [19]. The underlying SAT solver is called iteratively, updating the bounds for the number of satisfied soft clauses with each step. In this paper, we utilize the MaxSAT solver antom [20], applying this iterative approach.

B. Bounded Model Checking

Bounded Model Checking (BMC) is a technique which validates a sequential system by a given exploration limit up to a predefined number of time steps. In particular, BMC considers safety and liveness properties. A common application is to obtain error traces in sequential circuits which are required to falsify a certain safety property.

Intuitively, BMC starts from a fixed initial position (i. e., the set of initial states) and tries to attain a goal within a predefined maximum number of steps. First, it is validated whether the set of initial states already contains the goal. If this is not the case, BMC then checks whether the goal is reachable in one step, in two steps, etc. until either the goal is reached or the exploration limit is exceeded.

The structure of the system and the requested property are encoded as a propositional formula of the form:

$$BMC_k = I_0 \wedge T_{0,1} \wedge \ldots \wedge T_{k-1,k} \wedge P_k \tag{1}$$

I_0 encodes the initial states. The terms $T_{i,i+1}$ represent the so-called transfer function, which is the combinational part of the system. The transfer function defines one sequential step from time frame i to $i+1$ in the sequential system (e. g. a circuit) under consideration. The predicate P_k represents the goal, i. e., a property whose reachability after k steps has to be checked.

If there exists a path in the unfolded system starting at I_0 and reaching a state satisfying P_k in k time frames, BMC_k is satisfiable. In that case BMC returns a shortest witness satisfying the property. If the property never holds the BMC problem is unsatisfiable.

It can not be proven whether the property is never reachable unless the system is unfolded up to its diameter, and hence the procedure is not complete for all k less than the diameter (typically, the maximum bound for k is set far less then the system's diameter). In recent years some approaches have been presented which are able to show the unreachability of the property and therefore make BMC complete, namely *k-induction* [5] and *Craig interpolation* [6].

Craig interpolation uses the theorem of Craig interpolants [21], which represents an over-approximation of a particular set. Starting with the set of initial states, Craig interpolants are calculated for each transition step in order to obtain an over-approximation of the reachable state set. If the reachable state set does not change in two consecutive time steps a fix-point is reached and if the property is *not* part of the over-approximation it is proven that it is *never* reachable, and hence the BMC problem is unsatisfiable. If the property is part of the over-approximation, either the approximation was too coarse and the procedure is restarted excluding this spurious trace or we have shown that the goal is reachable. For more details of this procedure the interested reader is referred to [6]. In this paper, we make use of the complete BMC solver CIP [22] using Craig interpolants for proving whether the intermediate solution bounds represents a global optimum.

III. MaxBMC

In this section we formalize MaxBMC as an extension of BMC and state an algorithm to solve such kind of problem instances.

A. Definition

The MaxBMC problem is based on the BMC concept, i. e., it also asks whether a safety or liveness property in a sequential system holds within a predefined number of time steps. Additionally, if a witness is found, MaxBMC asks for the *quality* q of this witness. In particular, the witness with the highest quality for each transition step is identified, and therefore, MaxBMC determines the *Pareto-optimal* qualities for the sequence lengths until a bound k.

As in BMC, one can encode the structure of MaxBMC as CNF, including soft clauses:

$$MaxBMC_k = I_0 \wedge T_{0,1} \wedge \ldots \wedge T_{k-1,k} \wedge P_k \wedge O_k \tag{2}$$

The parts I_0, $T_{i,i+1}$ and P_k are defined as in Eq. 1. In extension to classical BMC we consider two properties for each unrolling depth. The first one is a property P_k describing requirements that need to hold in order to form a valid solution. The second objective O_k is a symbolic representation of any optimization problem which can be translated to MaxSAT. O_k consists of clauses that describe the *quality* of a witness (i. e., the soft clauses of the encoded MaxSAT problem) and provides the soft clause interface. Hence, the number of satisfied soft clauses are directly mapped to the quality.

Intuitively, we ask whether a property is reachable within a given bound k and if it is reachable we determine its quality. The quality is given by the MaxSAT objective of $MaxBMC_k$, i. e., the sum of the satisfied soft clauses in O_k. We denote the result of the optimization problem and therefore the quality of the best witness with sequence length k as q_k. The quality of witnesses with different lengths indicate *solution bounds*. Therefore we define a lower (q_{low}) and upper (q_{high}) bound of the optimization results q_i with $0 \leq i \leq k$.

As in BMC one can prove the reachability of the property in the same manner as described in Sec. II-B. Additionally, in MaxBMC we have to consider the case that for some depth i the property is reachable, but for a depth larger i it is not. This case is omitted in BMC, since one is only interested in the shortest trace to the good/bad state. This proof can be done quite similar to the standard reachability check by adding an additional constraint forcing a trace with at least depth $i+1$.

Furthermore, we extend the concept of proving reachability of a property to the question whether the solution bounds of the

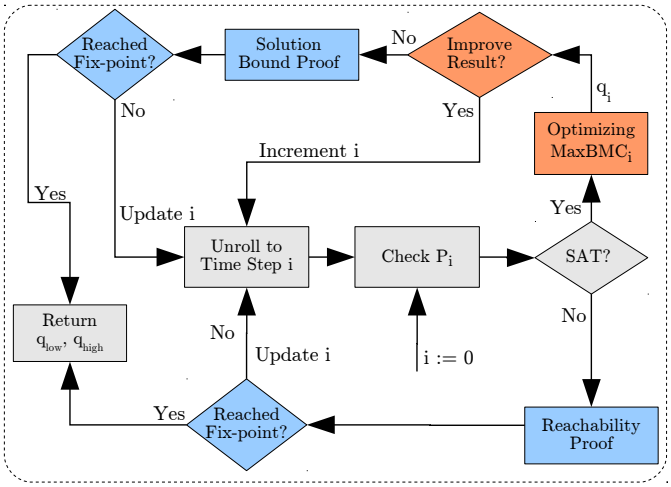

Figure 1. Algorithmic flow

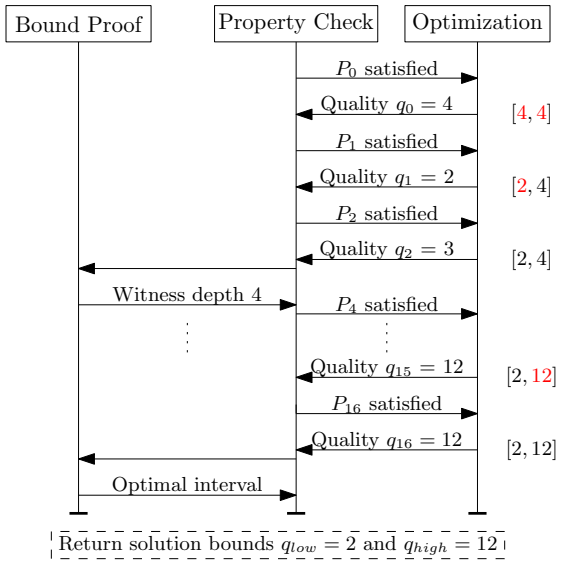

Figure 2. Illustrating MaxBMC example

secondary objectives can be extended. Therefore one has to apply a *solution bound proof*, demanding an optimization result q^* which is *not* within the solution bounds of q_{low} and q_{high}. If such a solution does not exist, we have shown the unreachability of a property, demanding an optimization quality q^* and hence obtained a proof that the solution bounds can not be improved. In the following sections we state more details of the composition of such a proof.

B. Algorithm

In this section we state the MaxBMC algorithm, which 1) computes the optimization results for each iteration and 2) proves the optimality of the solution bounds.

Fig. 1 shows the algorithmic flow, where the blue parts indicate steps involving a complete BMC solver, and the orange parts show the integration of a MaxSAT solver. An illustrating example is given in Fig. 2.

Validating $MaxBMC_k$ requires that the property P_k holds, which is always checked, before we can tackle the optimization problem. Starting with a depth of $i = 0$, it is checked whether P_i holds (in the initial states). In the example, the property P_0 is already satisfied, but in the general case it may not hold. In such situations we validate whether the property is (still) reachable beyond the current depth using a complete BMC solver. If the property does not hold the currently identified solution bounds are returned as no better solution can exist. Otherwise, a trace of minimal length to the next depth larger i where the property holds is generated. Hence, we update the depth i and continue with the corresponding time step.

At this point, a depth i is identified, where the property P_i holds and we determine the optimization objective for this depth using an iterative MaxSAT approach (c.f. II-A). We commit the BMC part of Eq. 2 as hard clauses and add the symbolic representation O_i to a MaxSAT solver. The solver returns the model from which we obtain a best witness for the current sequence length and its quality q_i. We compare the quality with the current lower and upper quality bounds $[q_{low}, q_{high}]$. In our example the quality of the first iteration is 4 and therefore the bounds are set to $[4, 4]$.

In case the bounds improved (as in the example from $[4, 4]$ to $[2, 4]$) the flow continues with incrementing the time step i, adding the next transition relation and checking the property. If there is no improvement for the solution bounds within a user-given number of time steps, we apply again a complete BMC solver to check the global optimality of the bounds. In the example, the quality does not change in the third iteration ($q_2 = 3$) and hence the solution bound proof is called (as the user-given threshold for non-improving time steps is set to 1 for this example).

To do so, we commit $MaxBMC_i$ as in Eq. 2 to the BMC solver, by adding the symbolic representation of O_i as part of the property P_i. In particular, the soft clauses are added together with encoding of the cardinality constraints, allowing to (de-)activate soft clauses as to be satisfied. Additionally, the constraint $(q^* < q_{low} \vee q^* > q_{high})$ is encoded and added in order to trigger a larger solution bound q^*.

We call this new combined property OP_i. If the BMC solver returns that OP_i is unreachable we can guarantee that q_{low} and q_{high} describe the maximum solution bounds and the flow terminates. Otherwise the solution bound proof returns a witness at a depth larger i which points to an improvement of the bounds. Consequently, i is updated and we continue with the check of P_i.

In the example the solution bound proof at depth 2 identifies a witness at depth 4 and the flow continues with checking P_4. When the proof is executed again at depth 16 (some steps are skipped in the example), the optimal bounds can be shown and the final solution bounds $q_{low} = 2$ and $q_{high} = 12$ are returned.

The algorithm is sound in the sense that 1) for every unrolling depth $i \leq k$ the optimization result q_i is calculated in case the property P_i holds and 2) the maximum solution bounds are given by q_{low} and q_{high}, if the optimality of the bounds could be proven.

The first part is easy to see: O_k is the representation of the quality as soft clauses. These clauses are added to a MaxSAT solver, whereas the remaining symbolic representation of $MaxBMC_k$ is added as hard clauses to the MaxSAT solver. Therefore the hard clauses are satisfied iff the property P_k holds. In that case the MaxSAT solver determines the number of satisfied soft clauses, which can be directly interpreted as the quality.

For the second part we need to validate that our solution bound proof is sound. Since we use an iterative MaxSAT approach for the optimization constraint, O_k is encoded as clauses which are added to the combined property. An accordant iterative MaxSAT solver would encode an additional network for the cardinality constraints for the soft clauses and iteratively add constraints bounding the optimization result via the network. Here, we add the network encoding as part of the property and bound the quality (i. e., the number of satisfied soft clauses) by triggering $< q_{low}$ or $> q_{high}$ implicitly as part of the combined property OP_k. Therefore, OP_k defines a safety property asking whether we reach a certain property (P_k) at which the optimization result is below $< q_{low}$ or above $> q_{high}$. The BMC solver will return such a witness with the shortest sequence length if it exists. Otherwise the BMC problem is classified as unsatisfiable.

The algorithm is complete in the sense that 1) it detects whether P_k is reachable and 2) returns the optimal solution bounds.

Again, the first part is clear, since a complete BMC solver checks whether P_k is reachable and therefore our algorithm is complete, too. This also holds in the case the property is reachable only until a time step i by forcing a solution with more than i time steps.

If the user-given bound k is large enough also the second part of the completeness holds. Since the solution bound proof is sound and the underlying procedure to check these bounds is complete, also the optimality of the solution bounds can be guaranteed.

C. Extensions

Based on the basic MaxBMC formalism defined in Eq. 2 one can define several natural extensions, depending on the specific requirements of an application.

One may extend the number of optimization objectives checked per time frame, i. e., there are possibly multiple O_k's in Eq. 2 that can be optimized together or separately. In the former case we obtain one combined quality q_k per sequence length k, in the latter case one has to solve multiple optimization problems per time step and consequently observe and prove multiple solution bounds.

Another straightforward extension is the definition of weights associated with each soft clause leading to *Weighted MaxBMC* likewise Weighted MaxSAT. In this case, the quality of a Weighted MaxBMC instance is given by the sum of the weights of the satisfied soft clauses in O_k.

IV. Implementation details

In this section we describe the underlying solver technologies we used and the modifications we made in order to solve the MaxBMC more efficiently.

A. Incremental MaxSAT

As shown in Section III, solving MaxBMC instances necessitates solving a series of similar MaxSAT problems. After each iteration, additional transition relations and further optimization constraints are added. Parts of earlier iterations are removed, but some stay unchanged and may still contain useful information. Likewise classical BMC with incremental SAT solving, MaxBMC may also benefit from an incremental MaxSAT solver which allows re-usage of information gained while solving previous MaxSAT instances. *Incremental MaxSAT* allows to change optimization constraints which necessitates adaptations to the solving process that go beyond classical incremental SAT.

The core of our incremental MaxSAT implementation is our in-house SAT solver antom [20], which supports incremental SAT solving and provides a MaxSAT interface. Internally, the maximization problem is expressed by a sorting network [23].

We modified the solver to include the following techniques known from the BMC context: *Constraint Sharing* [3] and *Constraint Replication* [4]. The reader is referred to the related references for more details of the methods.

Constraint Sharing and Constraint Replication are techniques for reusing learned clauses from former SAT solver calls. Using incremental solving in BMC, only one SAT instance, which changes with every call of the solver, is used for all calculations. In particular, parts of the formula are removed (e. g. by adding the transition relation $T_{i,i+1}$ and related property P_{i+1}, the property P_i has to be removed), and thus derived clauses originating from removed parts are invalid in later solver calls. In general, derived clauses from former solver calls are more beneficial for the BMC approach than using a single SAT instance for each transition step separately. In BMC the issues with invalid clauses in incremental solving are handled by adding *trigger* literals to the clauses which define the property.

The trigger literals allow to (de-)activate parts of the formula, which have to be removed in a later solver call. We call the clauses of these parts the *temporary clauses* of φ. Assume a trigger variable t_i for time frame i. Then each temporary clause$c = (l_1 \vee \ldots \vee l_n)$ from time frame i is extended to $c = (l_1 \vee \ldots \vee l_n \vee t_i)$. Thanks to the concept of assumption-based solving and the learning mechanism in modern SAT solvers all clauses which are derived by at least one temporary clause will also contain the trigger literal and hence are also marked as temporary clause. As an example in BMC consider a SAT solver call for transition step i. Adding the assumption $\neg t_i$ *activates* the temporary clauses (i. e. clauses describing the property P_i) of step i. At the same time the temporary clauses of the former transition step $i-1$ are *deactivated* by adding a clause containing only the trigger literal t_{i-1}. This will satisfy all temporary clauses with this trigger literal, and hence the property P_{i-1} is not constraining anymore.

The concept of trigger literals can be adapted likewise to incremental MaxSAT: the clauses of the properties P_k are (de-)activated as in BMC. Furthermore we have to consider clauses which are used for the encoding of the optimization constraints. We have to deactivate this encoding after each transition step, since the MaxSAT instance is only valid for the current one. Thus, the optimization constraints of transition step i have to be treated as temporary clauses for i. The trigger literal is added for these clauses in order to (de-)activate them.

Moreover, we have to take care of the constraints defining the optimization bounds added during a single MaxSAT solver call. If a classical iterative MaxSAT solver finds a bound for the optimization result, a new hard clause is added to the underlying SAT solver constraining this bound. Adding this bounding constriction is not sound anymore in an incremental approach since the bounds and any derived clauses originating from this constraint are only sound for the current transition step. Hence, instead of bounding the result by adding a clause, in incremental MaxSAT we add the bounds as assumption to the problem. This ensures that the constraint only holds for this time step and that any derived clause resulting from this constraint will contain the assumption literal, i. e., it is marked as a temporary clause and therefore we are able to deactivate these clauses.

In case q_k increases monotonically, i. e., the optimization result of a transition step is either equal or better than the result of the former ones, one can reuse these bounds. In particular, if we have determined an optimization result q_i in transition step i we can commit q_i as a minimum bound for all following transition steps to the MaxSAT solver. This can be done accordingly with monotonic decreasing q_k. In our experiments (c. f. Sec. VI) we observed that this extension is very beneficial for the MaxSAT solver. Typically, the optimization goal describes a mandatory part of the whole MaxSAT instance as the encoding of the cardinality constraints are quite expensive. This encoding can be largely simplified by adding the additional bounding constraints.

To the best of our knowledge, these extensions lead to the first *incremental MaxSAT solver*. The authors of [24] propose that the usage of incremental MaxSAT would be beneficial for their purpose, but it is presumably not implemented. By applying this solver, our algorithm is able utilize an incremental core algorithm as in BMC with incremental SAT solving and therefore profits from learned information to speed up the solving process.

B. Solution bound proof

We use the in-house BMC solver CIP [22], which supports unreachability proofs by Craig interpolation, as a back-end solver in order to derive on the one hand the proof whether the property P_k is reachable and on the other hand whether the solution bounds for the optimization problem can be improved. As described in Sec. III-B we have to encode the cardinality constraints of the MaxSAT problem. An additional constraint is added, demanding q^* original soft clauses to be satisfied, where q^* is either below q_{low} or above q_{high}. For example, consider the situation that we are currently applying time step i and q_{high} was not improved for a user defined threshold of time steps. Now, we call CIP, encoding the safety property: P_i holds and the quality of the optimization objective is higher than q_{high}. If there exists such a solution, CIP will return the witness with its sequential depth where the safety property is violated. Then we can update our result for q_{high} and proceed with our main MaxBMC loop. Otherwise, the BMC solver was able to prove that the quality is not above q_{high} for any sequential depth. Hence, we are able to fix q_{high} as the maximum optimization solution. This procedure is done analogously for q_{low}. If both q_{low} and q_{high} are fixed our algorithm terminates.

V. Applications

In this section we present applications that can be formalized as MaxBMC problems. First, we briefly introduce the problem of circuit initialization which also serves as a reference problem in the experimental results section. Second, we give some pointers to further applications which can be translated into our formalism.

A. Circuit initialization

The problem of *initializing* (or synchronizing) a given circuit is a well considered problem in the area of testing [10], [11] and is closely related to state reachability problems known from classical BMC. The problem handles the question whether a sequential circuit is initializable, i. e., all flip-flops can be set to a known value, starting from the completely unknown state. Since many circuits are not completely initializable due to their internal circuit function, there is a high interest in finding the maximal subset of flip-flops that can be initialized.

This problem can be formalized as a MaxBMC problem, where the number of initialized flip-flops can be seen as the quality of a trace, i. e., the optimization goal. To represent unknown values, the transfer function $T_{i,i+1}$ is encoded using $01X$ logic [25]. The maximization goal q_k is given by the number of flip-flops which can be initialized simultaneously. Hence, a maximization over these literals leads to the sequence that initializes as many flip-flops as possible. Note that the property P_k is always trivially true, since each solution to the underlying BMC instance represents a valid sequence even if no flip-flop gets initialized.

B. Further applications

A related problem to initialization sequence is the resetability of a sequential circuit, used for example in partial scan [26]. Since most circuits are not fully initializable, one may be interested in the minimum number of controlling flip-flops (i. e., flip-flops whose values must be controlled externally), leading to a fully initialized state. This objective can be applied naturally to MaxBMC by defining the number of non-controllable flip-flops as quality.

Another application area is the optimization of the dynamic power consumption of a sequential circuit [27] which can be estimated by the switching activity of circuit lines, i. e., the amount of times a certain line switches its logic value from 0 to 1 and vice versa. This switching activity can be defined as quality for MaxBMC. The resulting instance is able to identify traces in the system with minimum/maximum power consumption in order to optimize the design of the system.

Apart from the circuit domain there are extensions of the well-known planning problem: the question of cost-optimal planning or preference-based planning. In contrast to classical planning one seeks for secondary criteria for a plan (e. g., minimal sequence length, minimal action costs, or maximal state rewards). In [28], [29] methods are proposed to find such plans using Weighted MaxSAT in order to tackle the optimization criteria. The authors consider the construction of the plan detached from solving the MaxSAT problem and no incremental usage of a MaxSAT solver is applied. The plan construction can be formalized as satisfiability problem [30] and we suppose that these problems can be naturally translated into MaxBMC. A plan computation can be seen as traversing transition relations $T_{i,i+1}$, where the planning goal is a property P_k and the optimization criteria can be represented by different qualities q_k, translated into O_k.

In general our formalism covers problems containing a transition system together with cost functions, where the goal is to find a trace within the system with minimum/maximum costs. Additionally, we allow constricting properties, which have to hold independently from the cost-optimization goal.

VI. Experimental results

We considered the circuit initialization problem as described in Sec. V-A. Therefore we used sequential versions of commonly used academical and industrial benchmark circuits. All measurements were performed on a single core of a 3.3 GHz Intel Xeon processor with a time out of 4 CPU hours per circuit.

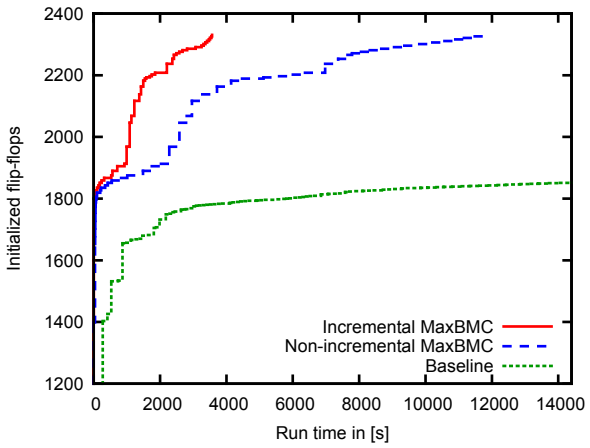

Figure 3. Method comparison for NXP benchmark circuit p45k-s

We implemented three versions for solving the MaxBMC problem to show the applicability of our incremental approach. The first one is a naïve implementation using only a complete BMC solver [31] ("Baseline"). We encode the optimization problem in CNF as part of the property demanding a successively increasing quality of the result until the complete BMC solver was able to prove that the result could not be increased. The second approach uses a MaxSAT solver and a complete BMC solver as described in Sec. III-B but without incremental MaxSAT solving ("Non-incremental MaxBMC"). The third implementation utilizes additionally the incremental techniques described in Sec. IV-A ("Incremental MaxBMC").

The exemplary comparison results are shown in detail in Fig. 3 for the NXP benchmark circuit "p45k-s". The figure shows the run time needed in relation to the number of initialized flip-flops for each method. As can be seen, the quality increases within time for each of the methods. However, the steepness of the individual curves differs greatly.

Although all methods were alloted the same timeout, the baseline method could only initialize 1848 flip-flops in 15 clock cycles (i. e. sequence length) while in the same time both MaxBMC approaches perform significantly better and could initialize the complete circuit (all 2331 flip-flops) within 49 clock cycles. However, the incremental approach reached the optimal results earlier. This general trend shown in the example can also be found for other circuits where the straightforward baseline approach yields only limited results due to the redundant calls to the BMC solver which can be avoided by our MaxBMC approaches. Moreover, the utilization of the incremental MaxSAT solver leads generally to further performance improvements.

Hence, we focus on the incremental MaxBMC variant in Table I. It shows the results for a broader range of challenging circuits which provide noteworthy results originating from the ISCAS 89 (named s*), ITC 99 (named b*) and industrial NXP (named p*k-s) benchmark series. The first three columns list the name of the circuit followed by the number of gates and flip-flop elements. The number of clock cycles of the best found sequence and the number of initialized flip-flops is given next. The column "Opt" indicates whether the result has been proven to be a global optimum. The last two columns show the run time, distinguished between the MaxSAT solver antom and the BMC solver CIP.

As can be seen, the MaxBMC algorithm is able to detect the initialization of a significant part of the circuits or prove that the number of initialized flip-flops can not be improved further. Exceptionally most benchmarks of the ITC 99 series can not be initialized at all, which is in general quickly detected by our reachability check. Whereas there are such sequences for ISCAS 89 and NXP circuits, which are either rather long and hence hard to derive (e. g. s38584, p78k-s) or the circuit structure is more complex and the solution bounds are therefore hard to prove (e. g. s38417, p267k-s). In the latter case we presume that the found bounds are already optimal but the BMC-solver is not able to

Table I
CIRCUIT INITIALIZATION OF ISCAS 89, ITC 99 AND NXP BENCHMARKS

			Best sequence			Run time in [s]	
Circuit	Gates	FF	Cycles	Init	Opt	antom	CIP
s05378	3221	179	10	179	1	0.73	0.00
s09234	6094	211	4	154	1	0.56	0.76
s13207	9441	638	19	303	0	9.94	>14390.06
s15850	11067	534	19	467	0	14.32	>14385.68
s35932	19876	1728	1	1728	1	0.39	0.00
s38417	25585	1636	9	372	0	44.33	>14516.70
s38584	22447	1426	36	1425	0	10831.60	>3568.40
b12	1127	121	77	48	1	45.06	138.08
b14	5923	245	0	0	1	0.05	1.84
b15	8026	449	0	0	1	0.08	5.50
b17	25719	1414	0	0	1	0.33	18.19
b18	76513	3270	0	0	1	1.05	176.03
b20	12991	490	0	0	1	0.11	4.60
b21	13168	490	0	0	1	0.12	5.47
b22	18789	703	0	0	1	0.17	3.82
p35k-s	48927	2173	3	2173	1	12.85	0.00
p45k-s	46075	2331	49	2331	1	3566.79	0.00
p77k-s	75033	3386	11	3386	1	77.57	0.00
p78k-s	80875	2977	32	2080	0	>14400	–
p100k-s	102443	5735	13	5020	0	>14400	–
p267k-s	296404	16528	8	1062	0	633.64	>13766.36
p330k-s	365492	16775	5	6596	0	>14400	–
p378k-s	404367	14885	11	3575	0	>14400	–
p469k-s	49771	332	1	3	1	2.86	3.05

derive the respective proof within the given time. But even if no optimal result was obtained, we were able to generate Pareto-optimal results for each depth up to the reached best sequence length, even for larger industrial circuits. Hence, our presented MaxBMC approach outperforms previous methods (e.g., [32], [33]) both in terms of scalability and quality.

VII. Conclusion

We presented MaxBMC, a formalism for defining optimization problems in sequential systems, optimizing secondary objectives. We state a sound and complete algorithm which is also able to yield the best possible solution bounds for each possible time step. Additionally, we developed an incremental MaxSAT approach by leveraging techniques from incremental SAT for BMC.

We express the problem of finding initialization sequences as MaxBMC and give some pointers to further applications. Experimental results demonstrate the effectiveness and applicability of our MaxBMC solver.

As future work we plan to investigate the usage of alternative incremental approaches for solving MaxSAT such as branch-and-bound and methods based on ILP solvers. This is of particular interest for solving Weighted MaxBMC where ILP formulations tends to be more beneficial than iterative MaxSAT approaches. Furthermore, we want to investigate in applying other techniques for providing the solution bound proofs.

Acknowledgements

The authors like to thank Stefan Kupferschmid for his insight in BMC and providing the CIP solver and Jürgen Schlöffel (Mentor Graphics Hamburg, formerly NXP) for supplying industrial benchmarks.

References

[1] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," in *Design Automation Conference*, pp. 317–320, ACM, 1999.

[2] J. Whittemore, J. Kim, and K. Sakallah, "SATIRE: A new incremental satisfiability engine," in *Design Automation Conference*, pp. 542–545, IEEE, 2001.

[3] N. Eén and N. Sörensson, "Temporal induction by incremental SAT solving," *Electronic Notes in Theoretical Computer Science*, vol. 89, no. 4, pp. 543–560, 2003.

[4] O. Strichman, "Accelerating bounded model checking of safety properties," *Formal Methods in System Design*, vol. 24, no. 1, pp. 5–24, 2004.

[5] M. Sheeran, S. Singh, and G. Stålmarck, "Checking safety properties using induction and a SAT-solver," in *Int'l Conf. on Formal Methods in Computer-Aided Design*, pp. 108–125, 2000.

[6] K. L. McMillan, "Interpolation and SAT-based model checking," in *Int'l Conference Computer Aided Verification*, pp. 1–13, 2003.

[7] E. Clarke, D. Kroening, and K. Yorav, "Behavioral consistency of C and verilog programs using bounded model checking," in *Design Automation Conference*, pp. 368–371, IEEE, 2003.

[8] D. G. Saab, J. A. Abraham, and V. M. Vedula, "Formal verification using bounded model checking: SAT versus sequential ATPG engines," in *VLSI Design*, pp. 243–248, IEEE, 2003.

[9] D. Große, U. Kuhne, and R. Drechsler, "Analyzing functional coverage in bounded model checking," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1305–1314, 2008.

[10] I. Pomeranz and S. M. Reddy, "On synchronizable circuits and their synchronizing sequences," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 9, pp. 1086–1092, 2000.

[11] M. Sauer, S. Reimer, S. M. Reddy, and B. Becker, "Efficient SAT-based circuit initialization for larger designs," in *VLSI Design*, 2014.

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

[13] A. Biere, "P{re,i}cosat@sc'09," in *SAT 2009 Competitive Event Booklet*, 2009.

[14] M. Soos, "Cryptominisat 2.5.0," in *SAT Race 2010 solver description*, 2010.

[15] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[16] P. Hansen and B. Jaumard, "Algorithms for the maximum satisfiability problem," *Computing*, vol. 44, no. 4, pp. 279–303, 1990.

[17] J. Marques-Silva and J. Planes, "Algorithms for maximum satisfiability using unsatisfiable cores," in *Advanced Techniques in Logic Synthesis, Optimizations and Applications*, pp. 171–182, Springer, 2011.

[18] H. Zhang, H. Shen, and F. Manya, "Exact algorithms for MAX-SAT," *Electronic Notes in Theoretical Computer Science*, vol. 86, no. 1, pp. 190–203, 2003.

[19] N. Eén and N. Sörensson, "Translating pseudo-Boolean constraints into SAT," in *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, pp. 1–26, 2006.

[20] T. Schubert and S. Reimer, "antom," in *https://projects.informatik.uni-freiburg.de/projects/antom*, 2013.

[21] W. Craig, "Linear reasoning: A new form of the Herbrand-Gentzen theorem," *Journal of Symbolic Logic*, pp. 250–268, 1957.

[22] S. Kupferschmid, M. Lewis, T. Schubert, and B. Becker, "Incremental preprocessing methods for use in BMC," *Formal Methods in System Design*, pp. 1–20, 2011.

[23] K. E. Batcher, "Sorting networks and their applications," in *AFIPS Spring Joint Computing Conference*, pp. 307–314, ACM, 1968.

[24] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani, "A modular approach to MaxSAT modulo theories," in *SAT*, vol. 7962 of *LNCS*, pp. 150–165, Springer, 2013.

[25] A. Jain, V. Boppana, R. Mukherjee, J. Jain, M. Fujita, and M. S. Hsiao, "Testing, verification, and diagnosis in the presence of unknowns," in *VLSI Test Symp.*, pp. 263–269, 2000.

[26] D. Lee and S. Reddy, "On determining scan flip-flops in partial-scan designs," in *Int'l Conf. on Computer-Aided Design. Digest of Technical Papers.*, pp. 322–325, 1990.

[27] K. Miyase, K. Noda, H. Ito, K. Hatayama, T. Aikyo, Y. Yamato, H. Furukawa, X. Wen, and S. Kajihara, "Effective IR-drop reduction in at-speed scan testing using distribution-controlling X-identification," in *Int'l Conf. on Computer-Aided Design*, pp. 52–58, IEEE, 2008.

[28] N. Robinson, C. Gretton, D. N. Pham, and A. Sattar, "Partial weighted MaxSAT for optimal planning," in *PRICAI 2010: Trends in Artificial Intelligence*, pp. 231–243, Springer, 2010.

[29] F. Juma, E. I. Hsu, and S. A. McIlraith, "Preference-based planning via MaxSAT," in *Advances in Artificial Intelligence*, pp. 109–120, Springer, 2012.

[30] H. A. Kautz and B. Selman, "Planning as satisfiability.," in *ECAI*, vol. 92, pp. 359–363, 1992.

[31] M. Sauer, S. Reimer, S. Kupferschmid, T. Schubert, P. Marin, and B. Becker, "Applying BMC, Craig interpolation and MAX-SAT to functional justification in sequential circuits," in *RCRA*, 2013.

[32] F. Corno, P. Prinetto, M. Rebaudengo, M. S. Reorda, and G. Squillero, "Initializability analysis of synchronous sequential circuits," *Transactions on Design Automation of Electronic Systems*, vol. 7, no. 2, pp. 249–264, 2002.

[33] Y.-L. Hou, M. Yang, R.-M. Zhao, and H. Lian, "Initialization for synchronous sequential circuits based on chaotic particle swarm optimization," in *Int'l Conf. on Machine Learning and Cybernetics*, vol. 3, pp. 1566–1571, 2010.