



Computing Over-Approximations with Bounded Model Checking

Daniel Kroening¹

*Computer Science Department
ETH Zürich
Switzerland*

Abstract

Bounded Model Checking (BMC) searches for counterexamples to a property ϕ with a bounded length k . If no such counterexample is found, k is increased. This process terminates when k exceeds the completeness threshold CT (i.e., k is sufficiently large to ensure that no counterexample exists) or when the SAT procedure exceeds its time or memory bounds. However, the completeness threshold is too large for most practical instances or too hard to compute. Hardware designers often modify their designs for better verification and testing results. This paper presents an automated technique based on cut-point insertion to obtain an over-approximation of the model that 1) preserves safety properties and 2) has a CT which is small enough to actually prove ϕ using BMC. The algorithm uses proof-based abstraction refinement to remove spurious counterexamples.

Keywords: Bounded model checking, SAT procedure, safety property

1 Introduction

In the hardware industry, formal verification is well established. Introduced in 1981, *Model Checking* [10,12] is one of the most commonly used formal verification techniques in a commercial setting. However, it suffers from the state explosion problem. In case of BDD-based symbolic model checking this problem manifests itself in the form of unmanageably large BDDs [7].

This problem is partly addressed by a formal verification technique called *Bounded Model Checking* (BMC) [6], introduced by Biere and others. In BMC,

¹ Email: daniel.kroening@inf.ethz.ch

the transition relation for a complex model M and its specification ϕ are jointly unwound up to a depth k to obtain a formula, which is then checked for satisfiability using a propositional SAT procedure such as Chaff [25]. In the case that ϕ is a safety property, the formula is satisfiable iff there exists a counterexample of length k , i.e., $M \not\models_k \phi$. If not so, k is increased to search for longer counterexamples. This process terminates either if the SAT procedure exceeds its time or memory bounds, a counterexample is found, or k exceeds a *completeness threshold* CT [17]. In the later case, k is sufficiently large to ensure that no counterexample exists, and thus, we conclude $M \models \phi$. BMC has been used successfully to find subtle errors in very large industrial circuits [27,14].

The disadvantage of BMC is that it is typically only applicable for refutation; the best known completeness threshold for properties of type $\mathbf{G}p$ is the *reachability diameter* of M , i.e., the longest shortest path from any initial state to any reachable state in the state graph. In practice, the diameter is usually too hard to compute, and furthermore, is often exponential in the number of state variables in the model. The *recurrence diameter* [6] is an over-approximation of the reachability diameter. However, it is still difficult to compute and typically much larger than the reachability diameter.

Thus, in practice, the principal method for *proving* safety properties is *abstraction*. Abstraction techniques reduce the state space by mapping the set of states of the actual, concrete system to an abstract, and smaller, set of states in a way that preserves the relevant behaviors of the system.

In the hardware domain, the most commonly used abstraction technique is *localization reduction* [19,28,8]. The abstract model \hat{M} is created from the given circuit by removing a large number of latches together with the logic required to compute their next state. The latches that are removed are called the *invisible latches*. The latches remaining in the abstract model are called *visible latches*. The initial abstract model is created by making the latches present in the property as visible, and the rest as invisible.

The abstract model is then passed to a model checker, typically BDD-based, such as SMV. Localization reduction is a *conservative* over-approximation of the original circuit for reachability properties. This implies that if the abstraction satisfies the property, the property also holds on the original circuit. The drawback of the conservative abstraction is that when model checking of the abstraction fails, it may produce a counterexample that does not correspond to any concrete counterexample. This is called a *spurious counterexample*.

In order to determine if the counterexample can be simulated on the concrete model, a Bounded Model Checking instance is typically formed: the

concrete transition relation for the design and the given property are jointly unwound to obtain a Boolean formula. The number of unwinding steps is given by the length of the abstract counterexample. The Boolean formula is then checked for satisfiability using a SAT procedure [28]. The transitions in the abstract trace are sometimes added to reduce the search space. The disadvantage is that other counterexamples of the same length may only be detected with additional refinement. If the instance is satisfiable, the counterexample is real and the algorithm terminates. If the instance is unsatisfiable, the abstract counterexample is spurious, and *abstraction refinement* has to be performed.

The basic idea of the abstraction refinement technique is to create a new abstract model which contains more detail (e.g., more visible latches) in order to prevent the spurious counterexample. This process is iterated until the property is either proved or disproved. There are numerous methods to refine the abstraction. If the abstract counterexample is used for refinement, the process is known as the *Counterexample Guided Abstraction Refinement* framework, or CEGAR for short [19,2,9,15,28].

Thus, successful application of abstraction refinement with localization reduction usually requires three components:

- (i) A BDD-based model checker that has enough capacity for the abstract model,
- (ii) a Bounded Model Checker with enough capacity to perform the simulation of the abstract trace,
- (iii) a way to refine the abstraction in case the simulation fails.

In practice, despite of the abstraction, the first step often turns out to be the bottleneck, especially if the property depends on many latches.

This paper proposes the use of a technique commonly applied by many hardware engineers in an informal and manual setting: If a design is too complex for either simulation or verification, engineers cut or partition the circuit. Formally, this corresponds to removing parts of the circuit and replacing the missing signals by non-deterministically chosen inputs. Such cut-points do not necessarily remove latches, and also may preserve logic dependent only on latches that were removed. The resulting circuit is an over-approximation of the original circuit with respect to safety properties.

Contribution

This paper proposes to use cut-point insertion [18] in order to compute an abstract model \hat{M} with two features: 1) \hat{M} over-approximates M , and thus, safety properties are preserved, and 2) we can syntactically (and thus, efficiently) identify a completeness threshold CT that is small enough to allow

BMC with bound \mathcal{CT} . Thus, if no counterexample is found, we can conclude $M \models \phi$. If a counterexample is found, we check if it is spurious. If so, the cut-points are refined in order to eliminate the spurious trace. Similar to [22], we use the proof of unsatisfiability of the failed simulation run for refinement.

We therefore can omit the BDD-based model checker in the abstraction refinement loop, and rely on BMC as the only reasoning engine. This allows proving many properties with BMC only.

Related Work

Baumgartner et al. [5] perform a structural analysis similar to the one used for this paper in order to obtain a completeness threshold. In contrast to the algorithm proposed in this paper, an abstraction of the circuit in order to obtain a smaller completeness threshold is not applied. The results are extended in [4].

The concept of the completeness threshold for BMC was introduced in [17]. A completeness threshold for arbitrary LTL properties is given in [11]. Optimizations to the diameter test that take the predicates in the property into account are given in [1].

Another popular technique to obtain a complete version of BMC is to use BMC to prove an inductive invariant [26]. The technique uses constraints to enforce simple (i.e., loop free) paths that are similar to the constraints used to perform recurrence diameter tests.

Somenzi et al. [20] use such constraints to obtain a complete BMC to be used on an abstract model in an abstraction refinement framework. As noted in [20], the depth that has to be searched using BMC can be exponentially larger than the reachability diameter.

Numerous methods have been proposed to refine an abstraction done by localization reduction. In [13], Clarke et al. propose the use of ILP solvers and machine learning techniques to choose a suitable set of latches for the abstract model. Details on how to improve the simulation step beyond the basic BMC instance are given in [3].

In [8], Chauhan et al. propose to analyze the conflict graph of the failed BMC run to obtain refinement information. A similar approach is used by McMillan [22]: the unsatisfiable core of the failed BMC run is analyzed to obtain the new set of latches used for localization reduction. The abstract model is verified using BDDs.

The first complete model checking approach based on SAT without any abstraction is presented by McMillan in [21]. A SAT solver is modified to perform pre-image computation. The approach enumerates states in the pre-image. Explicit state enumeration is avoided with an enlargement of the

assignment which is derived from the conflict graph.

In [23], McMillan presents the use of interpolants in order to obtain a complete model checker based on a BMC-like reasoning engine.

Outline

In section 2, we provide background information about bounded model checking, the completeness threshold, localization reduction, and automatic abstraction refinement. We describe the abstraction we apply in section 3. Experimental results are reported in section 4.

2 Background

2.1 The Completeness Threshold and the Diameter

Let M denote a finite transition system defined by a finite set of states S , a set of initial states $I \subseteq S$, and a transition relation $R \subseteq S \times S$. By $M \models \phi$ we denote that any computation of M satisfies the property ϕ , and by $M \models_k \phi$ we denote that all computations of length k or less do not violate ϕ .

Definition 2.1 The *Completeness Threshold* [17], denoted by \mathcal{CT} , for a finite transition system M and a property ϕ , is any natural number such that if there is no computation of length \mathcal{CT} that violates ϕ , ϕ holds for any computation done by M :

$$M \models_{\mathcal{CT}} \phi \longrightarrow M \models \phi$$

If $M \models \phi$, then the smallest such \mathcal{CT} is 0, and otherwise it is the length of the shortest counterexample. Thus, computing the smallest \mathcal{CT} is as hard as determining if $M \models \phi$ holds. In practice, one therefore aims at computing over-approximations of the smallest \mathcal{CT} .

Definition 2.2 The *Diameter* of a finite transition system M , denoted by $d(M)$, is the length of the longest shortest path (defined by its number of edges) between any two reachable states of M .

Definition 2.3 The *Initialized Diameter* of a finite transition system M , denoted by $d^I(M)$, is the length of the longest shortest path from any initial state to any reachable state of M .

It was already observed in [6] that $d(M)$ is a sufficiently large bound to prove properties of the form $\mathbf{AG}p$. This bound can be improved by using the initialized diameter $d^I(M)$. A bound for properties of the form $\mathbf{AF}p$ was identified in [17]. A method to compute a \mathcal{CT} for arbitrary LTL properties is found in [11].

Computing the Diameter

Testing if a particular k is the diameter corresponds to a QBF instance. Despite of the progress QBF solvers made, attempts to solve such instances have failed so far. Biere et al. suggested in [6] the use of SAT to compute the recurrence diameter, which is an over-approximation of the diameter. However, for most interesting circuits, the recurrence diameter is either too large or too hard to compute. Mneimneh and Sakallah [24] modify a SAT solver to compute the diameter by path enumeration.

2.2 Over-Approximating the Diameter with Structural Analysis

Model checking is frequently applied to circuits, which are typically given as a net-list. Baumgartner et al. [5] suggest to exploit the structure of these net-lists in order to compute an over-approximation of the diameter.

Definition 2.4 A *Net-list* is a directed graph (V, E, T) , where V is a finite set of vertices, $E \subseteq V \times V$ is the set of edges, and $T(v)$ is the type of the vertex $v \in V$. The type is one of **and** (AND-gate), **inv** (inverter), **reg** (register), **inp** (primary input). The in-degree of vertices of type **and** is at least one, of type **inv** and **reg** exactly one, and of type **inp** exactly zero.

Notation

Given two vertices v_1 and v_2 , we write $v_1 \xrightarrow{E} v_2$ iff $(v_1, v_2) \in E$, and $v_1 \rightsquigarrow^E v_2$ iff there is a path from v_1 to v_2 in E .

We write $v_1 \rightsquigarrow_G^E v_2$ if $T(v_1) = T(v_2) = \mathbf{reg}$ and there is a path from v_1 to v_2 in E that only goes through vertices (gates) of type $\{\mathbf{and}, \mathbf{inv}\}$. We require any such path to be acyclic, i.e., the logic between the latches must be combinational.

The definition of semantics for such a net-list is straight-forward. The conversion of circuits given in Verilog to such a net-list corresponds to synthesis.

Definition 2.5 The *Latch Dependency Graph* (LDG) of a net-list $N = (V', E', T)$ is a directed graph (V, E) , where $V = \{v \in V' \mid T(v) = \mathbf{reg}\}$ is the set of latches in N , and there is an edge between two latches v_1 and v_2 in the LDG iff there is a path from v_1 to v_2 in N that only uses gates, i.e., $v_1 \xrightarrow{E} v_2 \iff v_1 \rightsquigarrow_G^{E'} v_2$.

Definition 2.6 A *Component* inside a circuit is a connected subgraph of the LDG. The *Component Graph* is the graph generated by replacing each component by a single vertex.

We denote the bound we derive for the diameter of a component C by $\Delta(C)$. Obviously, 2^k is such a bound if k is the number of latches in C .

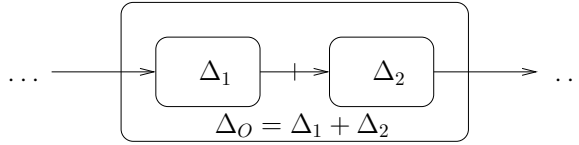


Fig. 1. Sequential composition of two components. A bound for the diameter of the composition is the sum of the individual diameters.

In [5], bounds for the diameter for various types of components are derived that are based on the structure of the component, e.g., for ROMs, constant latches, and acyclic components. In particular, it is observed that the sum of the bounds of the diameters of two components that are composed sequentially is a bound for the composition:

Theorem 2.7 *Let C_1 and C_2 be two components, and $\Delta(C_1)$ and $\Delta(C_2)$ be bounds for the diameter of C_1 and C_2 , respectively. The sum of the two bounds is a bound for the diameter of the sequential composition $C_1 \rightarrow C_2$ (Figure 1):*

$$\Delta(C_1 \rightarrow C_2) = \Delta(C_1) + \Delta(C_2)$$

2.3 Abstraction via Cut-Point Insertion

Cut-Point Insertion corresponds to replacing a signal in the net-list by a new primary input [18]. The resulting circuit \hat{M} is an over-approximation of the original circuit M , and a conservative abstraction for reachability properties. As already noted in [4], the completeness threshold of \hat{M} is *not* a completeness threshold for M ; the abstract circuit typically has a much smaller diameter. The diameter never increases by cut-point insertion.

3 A Complete BMC with Over-Approximation

3.1 Overview

Figure 2 shows an overview of the technique used in this paper. The algorithm follows the proof-based abstraction refinement loop used in [22]. We use cut-point insertion as described in section 2.3 as the abstraction technique. As initial abstraction, we insert cut-point such that all cycles in the net-list of the abstract model are eliminated. This results in a very small completeness threshold.

In contrast to most related papers that implement abstraction refinement, we do not use a BDD-based model checker to verify the abstract model \hat{M} . Instead, we compute a completeness threshold \mathcal{CT} of \hat{M} . This is described in detail in section 3.2. We then perform BMC on \hat{M} with bound \mathcal{CT} .

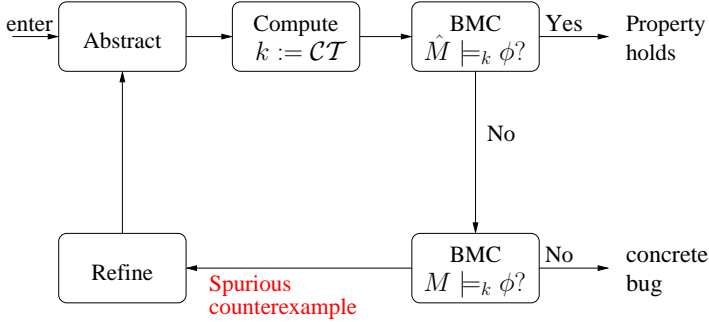


Fig. 2. Abstraction-refinement loop using BMC and the Completeness Threshold

If the property holds on \hat{M} , we can conclude it also holds on M , and the algorithm terminates. Otherwise, we obtain an abstract counterexample from the BMC run. The loop then proceeds as in the related work. The refinement step is slightly different and described in section 3.3.

3.2 Computing \mathcal{CT} in the Presence of Cycles

We extend the results introduced in [5] in order to obtain a completeness threshold for a larger class of designs. The main issue for the diameter over-approximation are cycles in the latch dependency graph. For cycle-free components, the most important results are summarized in section 2.2.

Thus, consider a circuit with cycles in the latch dependency graph. Such cycles are very common and typically arise from counters, or from forwarding logic in pipelined circuits. In order to over-approximate the diameter of such circuits, we define the concept of the *weighted* component graph.

Definition 3.1 The *weighted component graph* is a component graph (as in definition 2.6) in which a weight ω is assigned to each edge. We write $C_1 \rightarrow_\omega C_2$ iff there is an edge from C_1 to C_2 with weight ω . Let V_1 denote the set of latches in C_1 such that there is a path to a latch in C_2 in the LDG. Let V_2 denote these latches in C_2 . The weight corresponds to the number of signals that connect V_1 and V_2 .

As a special case, consider a circuit I with a diameter Δ_I . We assume that the circuit can be represented by a pipeline with $n := \Delta_I$ stages. Now add a single-bit feedback loop (Figure 3), which forms circuit O . The signal that forms the feedback loop is computed in the last stage of I and used as input for the first stage of I . There are arbitrary connections from stage i to stage $i + 1$, but no other connections are permitted.

Claim 3.2 The diameter of a simple pipeline pipeline with n stages and a single-bit feedback loop is bounded by $2 \cdot n$.

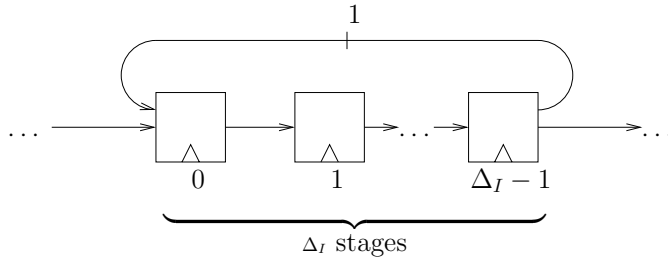


Fig. 3. The diameter of a component with a **one bit** self-loop is bounded by $2 \cdot \Delta_I$, where Δ_I denotes a bound on the diameter of the component without the loop.

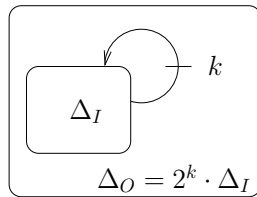


Fig. 4. The diameter of a component with self-loop is bounded by $\Delta_I \cdot 2^k$, where Δ_I denotes a bound on the diameter of the component without the loop, and k the weight of the loop.

We provide a proof of claim 3.2 in the appendix. This result can be generalized by eliminating the outer cycles bit by bit.

Claim 3.3 Let C_1, \dots, C_n denote a list of pair-wise different components that a) form a cycle in the dependency graph and b) contain no sub-cycle, i.e., $C_i \rightarrow C_j \iff j = i + 1 \vee (i, j) = (n, 1)$. The diameter of the component formed by this cycle is bounded by 2^k times the sum of the bounds of the diameters of the components, where k is the weight of any edge j on the cycle:

$$\Delta(C_1 \xrightarrow{\omega_1} C_2 \xrightarrow{\omega_2} \dots \xrightarrow{\omega_{n-1}} C_n \xrightarrow{\omega_n} C_1) = 2^{\omega_j} \cdot \sum_{i=1}^n \Delta(C_i)$$

The proof is done by re-arranging the components such that the desired edge represents the back-cycles and then by applying Claim 3.2 k times.

Example 3.4 If the cycle consists of one component only (Figure 4), the diameter of the component and the loop is bounded by $\Delta_I \cdot 2^k$, where Δ_I denotes a bound on the diameter of the component without the loop, and k denotes the weight of the loop. A k -bit counter is an example of this case.

If the cycle has more than one component, it is desirable to break the cycle on an edge with minimal weight, as the bound is exponential in the weight of the edge. This is depicted in Figure 5: The cycle can be removed using either edge.

Figure 6 shows two cycles that share a component. Such a cycle cannot be removed with the method above. The diameter is approximated using the number of latches in all components.

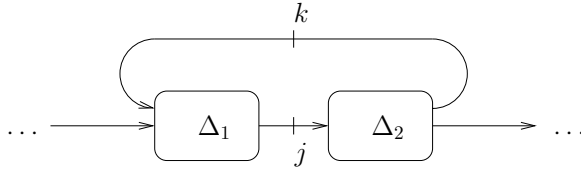


Fig. 5. The cycle can be broken using either the edge with weight k or j . The edge with minimal weight should be chosen.

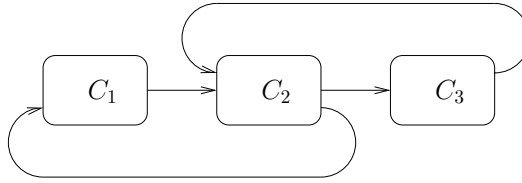


Fig. 6. Two cycles sharing a vertex C_2 . The cycles cannot be removed.

3.3 Refining the Abstraction

If a spurious counterexample is detected, we obtain the unsatisfiable core of the BMC instance used for the simulation. Similar to in [22], we identify the signals that are in this core (in [22], the latches are identified). We refine the cut-points by removing those cut-points that correspond to a signal found in the core. We do not introduce new cut-points.

4 Experimental Results

We have implemented the algorithm described in section 3. We make our implementation available for experimentation by other researchers.

We apply the algorithm to various circuits already used in [16] to determine its effectiveness. The benchmarks are taken from an implementation of an out-of-order RISC microprocessor with Tomasulo scheduler. We compare the performance of the new algorithm with the performance of plain Bounded Model Checking. All experiments are performed on an Intel Xenon machine with 2.5 GHz running Linux.

Bounded Model Checking is used for refutation only, i.e., it cannot conclude that there is no error trace. Instead, it checks the property up to a given number of cycles. In [16], the property checked was consistency with a C program. We check safety properties instead, which is easier. Table 1 summarizes the experimental results. A short description of each circuit can be found in [16].

In conclusion, traditional BMC typically outperforms the new algorithm if the property is to be refuted. This is to be expected, as refutation is done using a regular BMC instance in the refinement loop. However, the experiments also

Benchmark	latches	bug length	Run time BMC					Run time abstraction
			min.	10	20	30	40	
ALU_PIPE1	419	2	0.2s	3.5s	26.7s	132.5s	*	0.2s
ALU_PIPE2	419	-	-	107.7s	495.1s	*	*	1.1s
RF1	1024	-	-	7.4s	30.4s	56.3s	83.4s	7.0s
RF2	1024	1	0.4s	4.6s	7.8s	23.4s	*	7.5s
ROB1	2963	-	-	2.0s	5.4s	7.8s	22.1s	225.6s
ROB2	2963	-	-	182.8s	*	*	*	310.2s
ROB3	2963	16	1.8s	1.7s	4.2s	6.3s	8.7s	6.3s
ROB4	2963	64	*	4.3s	38.3s	124.0s	387.0s	33.3s

Table 1

Experimental Results. If no bug length is given, the property holds. The run time for BMC is given for various depths. The "min" column contains the run time for BMC for the shortest counterexample, if applicable. A star (*) denotes that the timeout of 1000s was exceeded.

show the benefit of the technique if the property is to be shown. In many cases, the refinement loop can show the property with a small bound.

5 Conclusion

We present an abstraction refinement loop that solely relies on BMC as its only reasoning engine. We use cut-point insertion in order to obtain an abstract model with a small completeness threshold. The completeness threshold is over-approximated with a structural analysis that permits cyclic circuits. If the abstraction is too coarse, cut-points are removed, which results in fewer spurious behavior but also a larger completeness threshold.

Our preliminary experimental results show that the technique performs well on circuits that implement a pipeline. We make our implementation available for experimentation by other researchers.²

Future Work

The algorithm presented in this paper requires severe restrictions of the shape of the latch dependency graph of the circuit. As future work, we plan to extend the algorithm that computes \mathcal{CT} in order to allow arbitrary latch dependency graphs. We also plan to improve the refinement algorithm such that cut-points are also added, not only removed.

² <http://www.inf.ethz.ch/personal/daniekro/ebmc/>

References

- [1] Awedh, M. and F. Somenzi, *Proving more properties with bounded model checking*, in: R. Alur and D. A. Peled, editors, *16th International Conference on Computer Aided Verification (CAV)*, Lecture Notes in Computer Science **3114** (2004), pp. 96–108.
- [2] Ball, T. and S. Rajamani, *Boolean programs: A model and process for software analysis*, Technical Report 2000-14, Microsoft Research (2000).
- [3] Barner, S., D. Geist and A. Gringauze, *Symbolic localization reduction with reconstruction layering and backtracking.*, in: E. Brinksma and K. G. Larsen, editors, *Proceedings of the 14th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science **2404** (2002), pp. 65–77.
- [4] Baumgartner, J. and A. Kuehlmann, *Enhanced diameter bounding via structural transformation*, in: *Design, Automation and Test in Europe Conference and Exposition (DATE)* (2004), pp. 36–41.
- [5] Baumgartner, J., A. Kuehlmann and J. A. Abraham, *Property checking via structural analysis*, in: E. Brinksma and K. G. Larsen, editors, *Proceedings of the 14th International Conference on Computer Aided Verification (CAV)*, Lecture Notes in Computer Science **2404** (2002), pp. 151–165.
- [6] Biere, A., A. Cimatti, E. Clarke and Y. Yhu, *Symbolic model checking without BDDs*, in: *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, Lecture Notes in Computer Science **1579** (1999), pp. 193–207.
- [7] Burch, J. R., E. M. Clarke, K. L. McMillan, D. L. Dill and L. J. Hwang, *Symbolic model checking: 10^{20} states and beyond*, *Information and Computation* **98** (1992), pp. 142–170.
- [8] Chauhan, P., E. M. Clarke, J. H. Kukula, S. Sapra, H. Veith and D. Wang, *Automated abstraction refinement for model checking large state spaces using sat based conflict analysis.*, in: M. Aagaard and J. W. O’Leary, editors, *4th International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, Lecture Notes in Computer Science **2517** (2002), pp. 33–51.
- [9] Clarke, E., O. Grumberg, S. Jha, Y. Lu and V. H., *Counterexample-guided abstraction refinement*, in: *CAV, 2000*, pp. 154–169.
- [10] Clarke, E., O. Grumberg and D. Peled, “*Model Checking.*” MIT Press, 1999.
- [11] Clarke, E., D. Kroening, O. Strichman and J. Ouaknine, *Completeness and complexity of bounded model checking*, in: *5th International Conference on Verification, Model Checking, and Abstract Interpretation*, Lecture Notes in Computer Science **2937** (2004), pp. 85–96.
- [12] Clarke, E. M. and E. A. Emerson, *Synthesis of synchronization skeletons for branching time temporal logic*, in: *Logic of Programs: Workshop*, LNCS **131**, Springer-Verlag, 1981 .
- [13] Clarke, E. M., A. Gupta, J. H. Kukula and O. Strichman, *SAT based abstraction-refinement using ILP and machine learning techniques.*, in: E. Brinksma and K. G. Larsen, editors, *Proceedings of the 14th International Conference on Computer Aided Verification (CAV)*, Lecture Notes in Computer Science **2404** (2002), pp. 265–279.
- [14] Coptly, F., L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella and M. Y. Vardi, *Benefits of bounded model checking at an industrial setting*, in: G. Berry, H. Comon and A. Finkel, editors, *Proceedings of the 13th International Conference on Computer Aided Verification (CAV 2001)*, number 2102 in Lecture Notes in Computer Science (2001), pp. 436–453.
- [15] Das, S. and D. Dill, *Successive approximation of abstract transition relations*, in: *16th Annual IEEE Symposium on Logic in Computer Science (LICS)* (2001), pp. 51–60.
- [16] Kroening, D. and E. Clarke, *Checking consistency of C and Verilog using predicate abstraction and induction*, in: *Proceedings of ICCAD* (2004), pp. 66–72.

- [17] Kroening, D. and O. Strichman, *Efficient computation of recurrence diameters*, in: L. Zuck, P. Attie, A. Cortesi and S. Mukhopadhyay, editors, *4th International Conference on Verification, Model Checking, and Abstract Interpretation*, Lecture Notes in Computer Science **2575** (2003), pp. 298–309.
- [18] Kuehlmann, A. and F. Krohm, *Equivalence checking using cuts and heaps*, in: *Proceedings of the 34th annual conference on Design automation (DAC)* (1997), pp. 263–268.
- [19] Kurshan, R., “Computer-aided verification of coordinating processes: the automata-theoretic approach,” Princeton University Press, 1994.
- [20] Li, B., C. Wang and F. Somenzi, *Abstraction refinement in symbolic model checking using satisfiability as the only decision procedure*, International Journal on Software Tools for Technology Transfer (STTT) **7** (2005), pp. 143–155.
- [21] McMillan, K., *Applying SAT methods in unbounded symbolic model checking*, in: *14th Conference on Computer Aided Verification*, 2002, pp. 250–264.
- [22] McMillan, K. and N. Amla, *Automatic abstraction without counterexamples*, in: *Proceedings of the 9th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, Lecture Notes in Computer Science (2003).
- [23] McMillan, K. L., *Interpolation and SAT-based model checking*, in: W. A. H. Jr. and F. Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verification (CAV)*, Lecture Notes in Computer Science **2725** (2003), pp. 1–13.
- [24] Mneimneh, M. and K. Sakallah, *SAT-based sequential depth computation*, in: *Proceedings of Asia South Pacific Design Automation Conference (ASPDAC)*, 2003, pp. 87–92.
- [25] Moskewicz, M., C. Madigan, Y. Zhao, L. Zhang and S. Malik, *Chaff: Engineering an efficient SAT solver*, in: *DAC*, 2001, pp. 530–535.
- [26] Sheeran, M., S. Singh and G. Stålmarck, *Checking safety properties using induction and a SAT-solver*, in: *Third International Conference on Formal Methods in Computer-Aided Design (FMCAD)* (2000), pp. 108–125.
- [27] Shtrichman, O., *Tuning SAT checkers for bounded model checking*, in: E. Emerson and A. Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*, Lecture Notes in Computer Science (2000), pp. 480–494.
- [28] Wang, D., P. Ho, J. Long, J. Kukula, Y. Zhu, T. Ma and R. Damiano, *Formal property verification by abstraction refinement with formal, simulation and hybrid engines*, in: *DAC*, 2001, pp. 35–40.

A Proofs

Let \mathcal{D}_i denote the range of values that the registers in stage i can take. Let $P_i(t) \in \mathcal{D}_i$ denote the value in pipeline stage i at time t . For $i > 0$, P_i only depends on P_{i-1} . Let f_i denote the function that represents this dependence:

$$(A.1) \quad P_i(t) = f_i(P_{i-1}(t-1))$$

Let $\Gamma(t) \in \mathbb{B}$ denote the value of the feedback bit at time t . The feedback bit is computed using P_{n-1} only. We use γ_{n-1} to denote the function that represents this dependency.

$$(A.2) \quad \Gamma(t) = \gamma_{n-1}(P_{n-1}(t))$$

This definition can be extended inductively to the other stages:

$$(A.3) \quad \gamma_i(x) := \gamma_{i+1}(f_{i+1}(x))$$

Thus, the data value $x \in \mathcal{D}_i$ in stage i will produce $\gamma_i(x)$ as feedback bit once it arrives in the last stage.

Definition A.1 We call $\gamma_i(x)$ the *color* of stage i .

Lemma A.2 For all $0 \leq i \leq j < n$, the color of stage j at time t is the color of stage i at time $t - j + i$:

$$\gamma_{n-1}(P_{n-1}(t)) = \gamma_i(P_i(t - j + i))$$

The proof of this lemma is easily done by induction on $n - 1 - i$.

Let $\iota(t)$ denote the value of the primary inputs in cycle t . The value computed for the first stage depends only on the feedback bit and these primary inputs. Let the value computed for the first stage be denoted by $f_0(\iota, \gamma) \in \mathcal{D}_0$.

Lemma A.3 The value in any stage at time $t \geq n$ only depends on primary inputs and on the color of the same stage n cycles earlier.

Proof. First, observe that $P_i(t)$ with $t \geq n$ only depends on the value of primary inputs during cycle $t - i - 1$ and on the value of the feedback bit at time $t - i - 1$:

$$(A.4) \quad P_i(t) = f_i \circ \dots \circ f_0(\iota(t - i - 1), \Gamma(t - i - 1))$$

By expanding the definition in Eq. A.2, we obtain:

$$(A.5) \quad P_i(t) = f_i \circ \dots \circ f_0(\iota(t - i - 1), \gamma_{n-1}(P_{n-1}(t - i - 1)))$$

We can use Lemma A.2 with $j = n - 1$ to rewrite Eq. A.5 and obtain:

$$(A.6) \quad P_i(t) = f_i \circ \dots \circ f_0(\iota(t - i - 1), \gamma_i(P_i(t - n)))$$

The color of stage i at time $t - n$ is $\gamma_i(P_i(t - n))$, which concludes the claim. \square

We now show the main claim 3.2.

Proof. [Claim 3.2] We show that we can bring the pipeline into any reachable state s within $2 \cdot n$ clock cycles or less.

If s is reachable, there must be a path s_0, \dots, s_t from an initial state s_0 to state $s = s_t$. Let t denote the length of the path. If $t \leq 2 \cdot n$, there is nothing to show.

Otherwise, we bring the circuit into state s as follows: (1) We start with the same initial state s_0 . (2) In the next n cycles, by picking appropriate primary inputs, we bring the pipeline into a state such that the colors at time n match the colors in state s_{t-n} . Such primary inputs exist, or otherwise, s_{t-n} is not reachable. (3) In cycles n to $2n - 1$, we bring the pipeline into the desired state by simply re-playing the primary inputs used to obtain s_{t-n+1}, \dots, s_t . This is sufficient according to lemma A.3. \square