Towards Formal Relaxed Equivalence Checking in Approximate Computing Methodology

Lukáš Holík, Ondřej Lengál, Adam Rogalewicz, Lukáš Sekanina, Zdeněk Vašíček, and Tomáš Vojnar
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence
Božetěchova 2, 61266 Brno, Czech Republic
Email: sekanina@fit.vutbr.cz

***Abstract*—Most design automation methods developed for approximate computing evaluate candidate solutions by applying a set of input vectors and measuring the error of the output vectors with respect to an exact solution. This approach is not, however, applicable when approximating complex combinational or sequential circuits since the error is not computed precisely enough. This paper surveys various methods of formal verification that could be extended for purposes of determining the error of approximation more precisely and formulates this task through a notion of formal relaxed equivalence checking.**

I. Introduction

In recent years, a new research field—*approximate computing*—was established to investigate how computer systems can be made better—more energy efficient, faster, and less complex—by relaxing the requirement that they are exactly correct. Approximate computing exploits the fact that some applications are inherently *error resilient*. Errors are not recognizable because human perception capabilities are limited (e.g., in multimedia applications), no golden solution is available for validation of the results (e.g., in data mining applications), or users are willing to accept some inaccuracies (e.g., when the battery of a mobile phone is almost depleted, but at least some basic functionality is still requested).

Therefore, the error (accuracy of computations) can be used as a design metric and traded for area on a chip, delay, throughput, or power consumption. Taking approximate computing closer to mainstream adoption requires a deeper understanding of inherent application resilience across a broader range of applications. After analyzing many applications, Chippa et al. [1] reported that about 83 % of runtime is spent in computations that can be approximated.

One way to reduce energy consumption by approximations is to allow timing errors. For example, turning down circuit voltage (at the expense of causing arithmetic errors every once in a while) or reducing the refresh rate of DRAM chips (at the expense of generating a few unwanted bit flips) are two possible approximation techniques. Another approximation technique is *functional approximation*. The idea of functional approximation is to implement a slightly different function to the original one provided that the error is acceptable and the power consumption or other system parameters are reduced adequately. The current literature describes various approximation methodologies. Two scenarios are dominating:

1) Ad hoc methods employed for approximation of a (single) particular component. For example, see the approaches proposed to approximate multipliers [2], adders [3], and median-outputting circuits [4].
2) Design automation methods developed for approximation of a class of problems (for example, circuit approximation methods SALSA [5], SASIMI [6], and ABACUS [7], as well as software approximation methods Chisel [8] and EnerJ [9]).

In the first scenario, a lot of knowledge about a particular system and its typical utilization can be incorporated into the approximation method. It is, however, difficult to apply the method for approximation of other systems.

In the second scenario, the approximations are performed using the same procedure for all problem instances of a given class. Approximate implementations showing different compromises between considered system parameters are generated and presented to the user, whose responsibility is to choose the most suitable approximate solution for a given application.

We will deal with the second scenario in this paper. All design automation methods based on functional approximations have to address two crucial issues:

- How to obtain useful approximations of the original implementation.
- How to assess the quality of the obtained approximations.

A functional approximation is typically obtained by a heuristic procedure that modifies the original, accurate (hardware or software) implementation. This heuristic procedure is repeated iteratively in order to improve the current approximate implementation in the subsequent steps. In each iteration, it is necessary to evaluate to what extent a given approximation satisfies functional and non-functional requirements (area, power consumption, etc.) of the specification. In some cases, any approximation satisfies functional requirements because the approximation is intentionally constructed with an error acceptable by the specification. This is, however, not typical in practice, where candidate approximations can show arbitrary errors and hence each approximate solution has to be evaluated to determine its functionality. The functionality is expressed using one or several error metrics such as the average error, error rate, and maximum error. After determining the error, non-functional properties of candidate approximations are also evaluated.

As the approximation heuristics typically work in many iterations, it is essential to quickly evaluate the quality of obtained approximations. Currently, the evaluation of the candidate approximations is usually done by applying a training data set and measuring the error (e.g., when image processing components are approximated). This approach is, however, not applicable in situations in which a complex entity has to be approximated and only a negligible error is acceptable (e.g., when a 32-bit adder is requested to correctly perform addition for 99.9 % of input combinations) because the training set needs to be too large (e.g., 2^{30} vectors).

Hence, there is currently a clear need to come up with a new approach to the problem of evaluating the quality of approximate complex digital systems (showing combinational as well as sequential behavior). Apart from a need of new suitable distance metrics, the crucial problem is to quickly calculate the distance between a candidate approximation and the exact implementation for complex problem instances. This problem will be called *relaxed equivalence checking* in the following, stressing the fact that the considered systems will be checked to be equal up to some bound w.r.t. a suitably chosen distance metric. For implementing relaxed equivalence checking, we propose building on advanced methods of *formal verification*.

The objective of this paper is to identify promising methods allowing one to implement fast and efficient relaxed equivalence checking algorithms, building on currently existing methods developed for (standard) equivalence checking and available approximation methodologies. Such algorithms would be very useful in approximation methods based on search algorithms.

The rest of the paper is organized as follows. Section II briefly surveys relevant methodologies and applications of approximate computing. Section III is devoted to relevant principles of formal verification. In Section IV, possible approaches to relaxed equivalence checking are discussed. Examples are introduced in Section V and conclusions are given in Section VI.

II. Approximate Computing

In approximate computing, the requirement on functional equivalence of the specification and implementation can be relaxed in order to not only reduce the power consumption but also accelerate computations or minimize the on-chip area. The concept has been developed in different ways and at various levels of the computing stack as discussed in the survey paper [11] and summarized below.

- *Ad hoc approximations in hardware.* Examples of approximate solutions include arithmetic circuits [2], [3], image processing components [2], pipeline circuits [12], fault tolerant circuits [13], or median circuits [4].
- *Design automation methods in hardware design.* Several design automation methods have been proposed to approximate hardware components. They are based on a heuristic procedure modifying the original circuit. Examples include: SALSA [5], MACACO [14], ASLAN[15] (for sequential circuits), SASIMI [6], [7], and genetic programming-based methods [16].
- *Software approximation and programming languages supporting approximate computing.* Artificial neural networks were proposed in [17] to approximate general-purpose code written in an imperative language. In the Chisel project, reliability- and accuracy-aware optimizations of computational kernels are performed by means of integer linear programming (ILP) and intended for approximate hardware platforms [8]. EnerJ [9] is an extension to Java that adds approximate data types and operations. Axilog is a set of language annotations that provide the necessary syntax and semantics for approximate hardware design and reuse in Verilog [18].
- *Specialized processors supporting approximate computing.* An integrated HW/SW approach to approximate computing has been developed in [19].

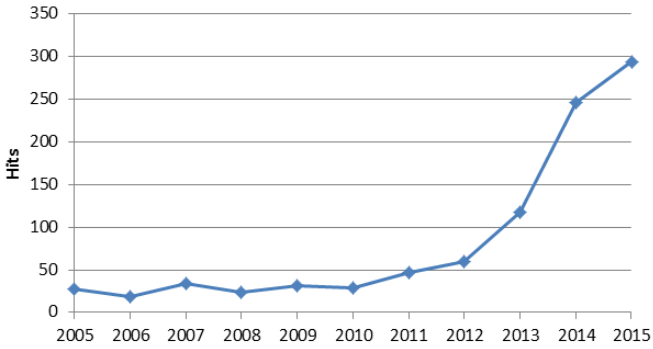

Fig. 1. The number of results for "approximate computing" by Google Scholar (November 2015).

The field of approximate computing is at an early stage of development, but a growing number of papers dealing with this topic indicates a very active research community (Fig. 1). An established design methodology is missing. Suitable benchmarking schemes have not been developed yet, which has, for example, the consequence that results of one paper are never compared with competitive approximate design methods presented in other papers!

III. Formal Verification

The task of comparing various aspects of the behaviour of different kinds of systems is crucial in the area of formal verification. The comparison can have the form of a top-level verification task as in the case of *systems equivalence checking*, or it can also be performed as one of many steps in the process of formal verification. The latter happens, e.g., within the so-called *fixpoint checks*, which are applied when iteratively computing the set of reachable states represented in some symbolic way. The fixpoint checks are used to see whether a newly computed set (represented symbolically) is bigger—i.e., contains some new reachable configurations—or equal to the previously computed one. This way, one can determine whether the state space exploration can be stopped or whether it has to continue.

In many cases, the objects to be compared in both of the above scenarios have the form of various kinds of *logical formulae* or *automata*—either given directly or derived from some source description of the considered systems (often using various heuristic techniques to cope with the involved *state space explosion* problem [20]). In the case of formulae, the comparison then boils down to checking their logical equivalence or entailment, while for the case of automata, it leads to checking their language equivalence or language inclusion.

A. Checking logical equivalence or entailment

In the area of efficient checking of equivalence or entailment in various logical fragments, a lot of progress has been achieved in the past years. This includes, for instance, the use of *binary decision diagrams* (BDDs) for compact representation and manipulation of propositional formulae [21]. Further, the recently quickly advancing technology of testing satisfiability of propositional formulae—the so-called *SAT solving* [22]—has been introduced. This technology allows one to efficiently check implications or equivalences of propositional formulae in many practical cases. For dealing with decidable first-order theories, one can then exploit the technology of *SMT solving* [23], which has also been advancing rapidly in recent years. Moreover, various efficient decision (or semi-decision) procedures or automated theorem provers have been proposed even for higher-order and/or undecidable logics or logical theories [24]. Furthermore, apart from SAT/SMT-solvers and other decision procedures, one can also use various *constraint solvers* from the domain of constraint programming [25].

B. Checking language equivalence or inclusion

The problem of checking language equivalence or inclusion is particularly difficult for various *non-deterministic automata*, regardless of whether they are used to accept finite or infinite words or trees (indeed, the inclusion problem is **PSPACE**-complete for words and **EXPTIME**-complete for trees). The reason is that non-deterministic automata cannot be easily complemented, which would reduce the inclusion problem to checking emptiness of the intersection of one of the original automata with the complement of the other. At the same time, non-deterministic automata do not only naturally arise within various iterative formal verification tasks [26], but they can be advantageous even for direct implementation in hardware [27] (their approximation is therefore an interesting topic itself).

A lot of effort has recently been invested into developing methods for efficient checking of equivalence or inclusion on non-deterministic automata. As a result, a number of new techniques based on the so-called antichains [28], [29] or bisimulations up to congruence [30] have emerged. Another obstacle when comparing languages of automata is dealing with *automata over infinite alphabets*, i.e., automata accepting words or trees whose symbols can be joined with data from some unbounded domain (such as the domain of integer or real numbers). This area, where the development has not been so extensive so far, has recently been targeted in a method that combines antichains, state-of-the-art techniques for dealing with data in formal verification, counter-example guided abstraction refinement, and Craig interpolation [31].

IV. Towards checking relaxed equivalence

Most formal verification approaches that build on testing *exact* equivalence, inclusion, or entailment are not directly extendable for relaxed equivalence checking. We believe, however, that the ideas behind efficient testing of exact equivalence/inclusion can serve as a basis for developing efficient methods for checking relaxed equivalence.

Moreover, we believe that these techniques can be extended to cope with various notions of relaxed equivalence suitable for search-based approximations, which themselves are still to be developed. For instance, in the case of combinational circuits, generic notions such as Hamming distance or various kinds of numerical distance, as well as application-specific relaxed equivalences (such as allowing the output bit vectors of two circuits to differ for the same input vectors only at the positions where the reference circuit outputs zero, which is useful in some fault-tolerant applications) have to be considered.

A. Relevant previous approaches

Instead of exact equivalence or inclusion, some works from the area of formal verification consider various notions of *simulation relations* [20]. These notions are, however, not directly applicable for the use in the context of approximate computing because they basically check whether one system is a refinement of another, not that one of the systems implements the other up to some number of errors. Nevertheless, suitable simulation relations could be used as an auxiliary tool to make the needed relaxed comparisons more efficient as in [29], [30].

A kind of relaxed equivalence checking that is closer to our aims appeared in [32], where a search method based on genetic programming was applied for evolution of mutual exclusion protocols. In this work, candidate solutions were compared through a number of temporal formulae describing desired properties of the target protocol. For each of these formulae, model checking was used to distinguish 4 quality levels ranging from “all executions are bad”, over “a bad execution can/cannot be extended to a good one” to “all executions are good”. This approach is, however, quite rough and rather specialised for the domain of reactive systems with a specification in the form of temporal formulae.

A form of relaxed equivalence checking appears in the work [33], which considers the *repair problem* for finite automata. The problem asks, given a pair of finite automata, what is their maximum *edit distance*, i.e., the number of edits needed to perform on a word from the language of one automaton to be in the language of the other automaton. This problem has also been studied in the context of weighted automata [34]. These works are, however, mostly complexity-theoretic ones not aiming at truly efficient algorithms to be used for the problem (e.g., when computing the distance of two automata, all possible bounds of some polynomial size are

checked in [33], with each of the tests being **PSPACE**-hard). Moreover, the works consider just a single distance metric. It would be interesting to investigate other possible distance metrics as well (measuring, e.g., the average number of needed edits, compare the particular alphabet symbols in a finer way, not just whether they are the same or not, etc.).

As for checking relaxed equivalence of Boolean functions, one promising approach seems to be to compute the Hamming distance of the functions, i.e., the number of Boolean assignments that are models of one of the formulae but not of the other. This can be reduced to the problem of counting the models of the Boolean formula obtained from the two functions using XOR, i.e., to the so-called #SAT problem (which is a well-known **#P**-complete problem). This should enable a use of the so-called #SAT-solvers [35].

An orthogonal direction to checking relaxed equivalence is *approximation* of *exact equivalence*. In the context of search heuristics, methods that quickly give a good fitness approximation are often preferred to slower exact ones. For propositional formulae, there has been proposed an approximation algorithm for the #SAT problem in [36]. An approximation of testing equivalence of finite automata appeared in the context of their learning using the probabilistically approximately correct (PAC) approach [37]. In this setting, language equivalence is approximated by checking membership of randomly sampled inputs. Another approach to approximate testing of automata equivalence emerged in the scope of probabilistic automata [38]. The approach reduces the equivalence problem to testing identity of polynomials which can be done with an arbitrary precision using a randomised algorithm in polynomial time.

B. Practical aspects

Consider an exact solution E_0 and its approximation E_1 created by means of a search operator. In order to determine their distance under a given metric, it is often more efficient to extract parts in which E_0 and E_1 differ and calculate the distance using only these different parts rather than using complete E_0 and E_1.

There is a possibility that the search operator could be constructed in such a way that the parts in which E_0 and E_1 differ are very small and thus easy for relaxed equivalence checking. Hence the algorithm developed for relaxed equivalence checking and the search method should be designed together.

In some cases, the quality of E_1 is so poor that its unsuitability can easily be recognized by applying a relatively small set of input vectors. If this technique is supported, the evaluation procedure can be speeded up because a complex (relaxed) equivalence checking solver is not called.

The error is not the only objective criterion of the approximation procedure. Power consumption, performance (delay), area on a chip (in the case of circuits), and other parameters have to be determined for candidate approximations. As some hard constrains on all these parameters are usually given (for example, the error must be less than 1 % and the minimum operation frequency is 100 MHz), the parameters of candidate solutions should be evaluated according to the time needed for their obtaining (i.e., the least time requiring parameter has to be evaluated first). For example, if the maximum operation frequency of a circuit can be obtained very quickly, but obtaining the error is very time consuming, then the maximum frequency has to be determined firstly in order to skip the (possibly useless) error computation in the case that the obtained frequency is violating the constraints and hence the candidate approximation must be rejected anyway. A smart evaluation procedure can save a lot of valuable time.

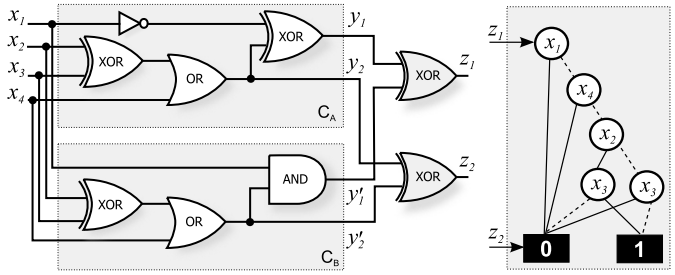

Fig. 2. Determining the Hamming distance of two combinational circuits using a BDD.

V. Examples

This section presents two available methods for combinational circuit approximation in which relaxed equivalence checking has already been included.

A. BDD-based approach

The error quantification is usually based on an arithmetic error metric in existing approximation methods. In order to approximate general logic (such as pattern matching circuits and complex encoders) in which no additional information is usually available to establish a suitable error metric, a genetic programming-based method capable of approximating combinational circuits under the Hamming distance as a metric was developed in [39]. The Hamming distance between the vectors produced by exact circuit and approximate circuit was determined using BDDs as follows.

Let us suppose that both circuits have k inputs denoted $x_1 \dots x_k$ and m outputs denoted $y_1 \dots y_m$ and $y'_1 \dots y'_m$, respectively. Corresponding primary inputs of both circuits are aligned and corresponding primary outputs y_i and y'_i are connected using the XOR gates. The goal is to obtain one circuit with k primary inputs $x_1 \dots x_k$ and m primary outputs $z_1 \dots z_m$, $z_i = y_i$ XOR y'_i. In order to disprove the equivalence, it is then sufficient to identify at least one output z_i whose $Onset(z_i)$ is not empty, i.e., to find an input assignment x for which the corresponding outputs y_i and y'_i provide different values (Fig. 2).

The Hamming distance between the truth tables of these circuits can be obtained by applying the *Sat-Count* function on every output z_i and summing the results. In the example shown in Fig. 2, *Sat-Count* will return 2 for z_1 and 0 for z_2, i.e., the Hamming distance is $0 + 2 = 2$. It can easily be

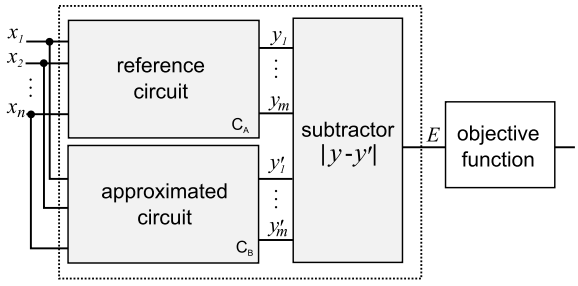

Fig. 3. Auxiliary circuit for SAT-based worst-case error analysis

checked that if $x \in \{0000, 0110\}$, the circuits provide different output values.

Various trade-offs between the average Hamming distance, area, and delay can be obtained using this method for circuits containing hundreds of gates and tens of inputs and outputs.

B. SAT-based approach

In order to check whether a predefined worst-case error is violated by the candidate approximate circuit, a pseudo-Boolean SAT solver combining a SAT solver with an ILP solver was employed in [14]. The principle of the method is shown in Fig. 3. Firstly, an auxiliary circuit is constructed. This circuit instantiates the candidate approximate circuit and the accurate (reference) circuit and compares their outputs to quantify the error for any given input. Then, the auxiliary circuit is converted to a CNF formula and the resulting formula is used together with an objective function as input of the SAT solver. The objective function is constructed in such a way that it maximizes the value at the output E.

However, while violating the worst error can be detected, no method capable of establishing, for example, the average error using a SAT solver has been proposed up to now.

VI. Conclusions

Contrasted to the evaluation of functionality of a candidate approximate solution using a set of vectors, we surveyed relevant methods of formal verification that could provide more exact information about the error. We have developed a notion of formal relaxed equivalence checking and surveyed relevant methods developed in the field of formal verification that could be used as a basis for efficient relaxed equivalence checking.

It turns out that only a few equivalence checking methods can be almost directly extended for purposes of relaxed equivalence checking. These methods are, however, not suitable for determining key distances such as the average error. Another problem is limited scalability of equivalence checking, which is currently applicable only to small and middle-sized systems.

Our future research will be devoted to developing a suitable formal relaxed equivalence checking method together with an efficient search algorithm in order to quickly compute the error of candidate approximations under a predefined error metric and thus accelerate the approximation procedure for complex problem instances.

VII. Acknowledgments

This work was supported by the Czech Science Foundation project 16-17538S (Relaxed equivalence checking for approximate computing).

References

[1] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *The 50th Annual Design Automation Conference 2013, DAC'13*. ACM, 2013, pp. 1–9.

[2] P. Kulkarni, P. Gupta, and M. D. Ercegovac, "Trading accuracy for power in a multiplier architecture," *J. Low Power Electronics*, vol. 7, no. 4, pp. 490–501, 2011.

[3] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124–137, 2013.

[4] M. Monajati, S. M. Fakhraie, and E. Kabir, "Approximate arithmetic for low-power image median filtering," *Circuits, Systems, and Signal Processing*, vol. 34, no. 10, pp. 3191–3219, 2015.

[5] S. Venkataramani, A. Sabne, V. J. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: systematic logic synthesis of approximate circuits," in *The 49th Annual Design Automation Conference 2012, DAC'12*. ACM, 2012, pp. 796–801.

[6] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits," in *Design, Automation and Test in Europe, DATE'13*. EDA Consortium San Jose, CA, USA, 2013, pp. 1367–1372.

[7] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, "ABACUS: A technique for automated behavioral synthesis of approximate computing circuits," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE'14. EDA Consortium, 2014, pp. 1–6.

[8] S. Misailovic, M. Carbin, S. Achour, Z. Qi, and M. C. Rinard, "Chisel: Reliability- and accuracy-aware optimization of approximate computational kernels," in *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*. ACM, 2014, pp. 309–328.

[9] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate data types for safe and general low-power computation," in *Proc. of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2011, pp. 164–174.

[10] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," *Commun. ACM*, vol. 58, no. 1, pp. 105–115, Dec. 2014.

[11] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proc. of the 18th IEEE European Test Symposium*. IEEE, 2013, pp. 1–6.

[12] S.-L. Lu, "Speeding up processing with approximation circuits," *IEEE Computer*, vol. 37, no. 3, pp. 67–73, 2004.

[13] M. Choudhury and K. Mohanram, "Low cost concurrent error masking using approximate logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 8, pp. 1163–1176, 2013.

[14] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "MACACO: Modeling and analysis of circuits for approximate computing," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2011, pp. 667–673.

[15] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, "ASLAN: Synthesis of approximate sequential circuits," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE'14. EDA Consortium, 2014, pp. 1–6.

[16] Z. Vasicek and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 432–444, 2015.

[17] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Proc. of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2012, pp. 449–460.

[18] A. Yazdanbakhsh, D. Mahajan, B. Thwaites, J. Park, A. Nagendrakumar, S. Sethuraman, K. Ramkrishnan, N. Ravindran, R. Jariwala, A. Rahimi, H. Esmailzadeh, and K. Bazargan, "Axilog: Language support for approximate hardware design," in *Design, Automation and Test in Europe, DATE'15*. EDA Consortium, 2015, pp. 1–6.

[19] V. Chippa, S. Venkataramani, S. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing: An integrated hardware approach," in *2013 Asilomar Conference on Signals, Systems and Computers*. IEEE, 2013, pp. 111–117.

[20] A. Valmari, "The state explosion problem," in *Lectures on Petri Nets I, Basic Models*, ser. LNCS, vol. 1491. Springer, 1996.

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

[22] O. Ohrimenko, P. J. Stuckey, and M. Codish, "Propagation = lazy clause generation," in *Principles and Practice of Constraint Programming*, ser. LNCS, vol. 4741. Springer, 2007.

[23] R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T)," *Journal of the ACM*, vol. 53, no. 6, pp. 937–977, 2006.

[24] M. Fitting, *First-Order Logic and Automated Theorem Proving (2nd ed.)*. Springer, 1996.

[25] S. Minton, A. Philips, M. D. Johnston, and P. Laird, "Minimizing conflicts: A heuristic repair method for constraint-satisfaction and scheduling problems," *Journal of Artificial Intelligence Research*, vol. 58, no. 1–3, 1993.

[26] P. Habermehl, L. Holík, A. Rogalewicz, J. Šimáček, and T. Vojnar, "Forest automata for verification of heap manipulation," *Formal Methods in System Design*, vol. 14, no. 1, pp. 83–106, 2012.

[27] R. Sidhu and V. K. Prasanna, "Fast regular expression matching using FPGAs," in *Proc. of FCCM'01*. IEEE, 2001.

[28] M. D. Wulf, L. Doyen, T. A. Henzinger, and J. Raskin, "Antichains: A new algorithm for checking universality of finite automata," in *Proc. of CAV'06*, ser. LNCS, vol. 4144. Springer, 2006.

[29] P. Abdulla, Y.-F. Chen, L. Holik, R. Mayr, and T. Vojnar, "When simulation meets antichains," in *Proc. of TACAS'10*, ser. LNCS, vol. 6015. Springer, 2010, pp. 158–174.

[30] F. Bonchi and D. Pous, "Checking NFA equivalence with bisimulations up to congruence," in *Proc. of POPL'13*. ACM, 2013.

[31] R. Iosif, A. Rogalewicz, and T. Vojnar, "Abstraction refinement for trace inclusion of data automata," *CoRR*, vol. abs/1410.5056, 2014. [Online]. Available: http://arxiv.org/abs/1410.5056

[32] G. Katz and D. Peled, "Genetic programming and model checking: Synthesizing new mutual exclusion algorithms," in *Proc. of ATVA'08*, ser. LNCS, vol. 5311. Springer, 2008.

[33] M. Benedikt, G. Puppis, and C. Riveros, "Regular repair of specifications," in *Proc. of LICS'11*. IEEE, 2011.

[34] M. Mohri, "Edit-distance of weighted automata: General definitions and algorithms," *International Journal of Foundations of Computer Science*, vol. 14, no. 6, 2013.

[35] M. Thurley, "SharpSAT—counting models with advanced component caching and implicit BCP," in *Proc. of SAT'06*, ser. LNCS, vol. 4121. Springer, 2006.

[36] M. Thurley, "An approximation algorithm for #k-SAT," in *Proc. of STACS'12*, ser. LIPIcs, vol. 14. Schloss Dagstuhl, 2012.

[37] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, no. 2, pp. 87–106, 1987.

[38] S. Kiefer, A. S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell, "Language equivalence for probabilistic automata," in *Proc. of CAV'11*, ser. LNCS, vol. 3806. Springer, 2011.

[39] Z. Vasicek and L. Sekanina, "Evolutionary approximation of complex digital circuits," in *Genetic and Evolutionary Computing Conference*. ACM, 2015, pp. 1505–1506.