SAT and ATPG: Boolean engines for formal hardware verification

Armin Biere

Dept. of Computer Science
ETH, Zürich, Switzerland

Wolfgang Kunz

Dept. of Electrical Engineering
University of Kaiserslautern, Germany

Abstract

In this survey, we outline basic SAT- and ATPG-procedures as well as their applications in formal hardware verification. We attempt to give the reader a trace trough literature and provide a basic orientation concerning the problem formulations and known approaches in this active field of research.

1 Background

Checking *satisfiability* (SAT) of propositional formulae in *conjunctive normal form* (CNF) is the classical NP-complete problem. A formula is in CNF if it is a product of sums of literals. Many hard problems can be translated into a SAT problem. This has been the main motivation to work on good heuristics and algorithms. The idea is that once implemented in a generic SAT-solver good heuristics could be used and shared across multiple application domains. Additionally, the abstract framework often allows to find general heuristics more easily than it would be possible with a narrow application point of view.

The research around SAT-procedures started in the context of automated theorem proving, where SAT was identified as a simple instance of formally proving theorems. Theorem proving is also regarded as a subfield of artificial intelligence. Most of this work is reviewed in [KB99]. In the last decade many advances in SAT were driven by the electronic design automation (EDA) community with their huge interest in efficiently solving large SAT problems.

Research in *automatic test pattern generation* (ATPG) on the other hand was primarily driven by specific applications in circuit testing. It is the task of an ATPG-algorithm to generate a test for every fault in the circuit according to some fault model. If the well-known *stuck-at fault model* is assumed a test is obtained by finding a set of input assignments such that the fault is *controlled* at the fault location, i.e., a ´0´ is produced for a stuck-at-1 and a ´1´ is produced for a stuck-at-0, and the fault is *observable*, i.e., a signal change at the fault location propagates to at least one ouput of the circuit. We note that the problem formulations of SAT and ATPG are closely related. The controllability portion of the ATPG problem immediately represents a SAT problem for a given signal (or its inversion) in a gate netlist. Observability can be mapped to Boolean satisfiability as well by using the notion of *Boolean difference* [AB90]. ATPG for single stuck-at faults can therefore be solved by a SAT-solver as was shown in [La89, SB96]. Conversely, SAT problems can be solved by ATPG tools, if the CNF is interpreted as a two-level circuit description. In spite of these similarities, however, there are important differences between SAT and ATPG that result from the fact that SAT-algorithms operate on a CNF while conventional ATPG-algorithms operate on a multi-level Boolean network. This difference along with the different historic background has lead to a fairly different terminology in the SAT and ATPG literature.

2 Converting a gate netlist into CNF

Early usage of SAT in EDA took place in the context of minimizing two-level logic [BH84]. More generally, when dealing with multi-level circuits, it is important to have an efficient translation of a circuit structure into CNF. This was first addressed in [Ts70]. The Tseitin translation introduces new literals for all circuit nodes and generates relational constraints in CNF to capture the functional behaviour of each gate.

As an example for the Tseitin construction consider the circuit with three inputs a, b, c, the output o and the sum of products representation $o = a + bc$. We introduce a new variable t for the AND gate representing the conjunction of b and c. The relation between the output o and the input can then be described with the two equations $o = a + t$ and $t = bc$. Now we can translate these equations into six implications: $o \rightarrow (a + t)$, $a \rightarrow o$, and $t \rightarrow o$ for the first equation and $t \rightarrow b$, $t \rightarrow c$, and $(bc) \rightarrow t$ for the second equation. The resulting CNF is obtained by replacing implications by disjunctions:

$$(o' + a + t)(a' + o)(t' + o)(t' + b)(t' + c)(b' + c' + t)$$

Even for more complicated gates, such as gates with multiple inputs or XOR gates, the translation of the generated equalities into implications and then into CNF is pretty straightforward. If we restrict *counting* gates like XOR to have a constant maximal fanin, then the whole translation remains linear in the number of gates.

3 ATPG versus SAT procedures

Both, modern SAT- and ATPG-algorithms approach the decision problem by a backtrack search in the finite Boolean space that is spanned by the variables of the CNF or the Boolean network, respectively. In SAT, this was first formulated in the *Davis-Logeman-Loveland* (DLL) procedure [DL62] and in VLSI testing the famous *D-algorithm* [Ro66] was the first complete test generation algorithm. Note that not DLL but the *Davis-Putnam*

0-7803-7607-2/02/$17.00

procedure [DP60] is the first complete SAT-solving method. The Davis-Putnam procedure is based on *resolution* which leads to a fundamentally different reasoning scheme compared to backtrack search. Almost all modern solvers are based on backtrack search, however. They exhaust a binary decision tree and employ resolution only as an optional instrument to prune the search space. The first ATPG tool enumerating a binary decision tree is PODEM [Go81]. Efficient heuristics to prune the search space have further been proposed in [FS83, KM87, GB91, GF01]. Later work in ATPG mainly concentrated on effective implication procedures [SA89, RC90, KP92, CA93, TG00, GF01]. Implications are needed to determine necessary assignments in the search process and help to avoid backtracks. Efficient implication routines are also key in SAT. In the SAT terminology making implications is referred to as *Boolean constraint propagation* (BCP) and many notions of BCP can be related to concepts of implication techniques in ATPG [Ha02].

In SAT procedures, the CNF representation facilitates powerful methods to prune the search space based on conflict analysis [MS99, BS97, Zh97]. When conflicts occur clauses are added to the clause base and help to avoid these and related conflicts in the future. Such clause recording is quite specific for CNF-based solvers and has been explored only little for ATPG algorithms operating on multi-level gate netlists. Conflict analysis is often combined with *non-chronological backtracking*. Non-chronological backtracking deviates from the rigorous scheme of the binary decision tree by analyzing the true causes of a conflict using an implication graph [MW85, Ma86, MS99]. Recent work, incorporates a special form of non-chronological backtracking combined with conflict analysis in an ATPG framework based on the D-algorithm [WR02].

The main differences between SAT and ATPG arise from the slightly different problem formulation and the different representation of the problem as CNF or multi-level netlist. In ATPG-based logic synthesis (see the following paper in these proceedings) the observability part of the ATPG problem is crucial. In many other EDA applications, however, observability is of little relevance and only the controllability part of ATPG is used. With the notable exception of [Br93], e.g., this is true for applications in equivalence checking and property checking. ATPG-algorithms that only solve the controllability problem and operate on a multi-level gate netlist are sometimes called *structural SAT-solvers*. This terminology relates to the important issue of choosing an appropriate representation of the problem. A representation as structural gate netlist facilitates many heuristics that exploit structural circuit information and takes into account the multi-level nature of circuits. A CNF-representation, on the other hand, has the advantage that logic relationships and constraints can be represented very efficiently as clauses. Also, a CNF is very regular so that efficient data structures can be developed. Especially, in CNF-based SAT-solving a lot of progress has been contributed by engineering efficient data structures for BCP [Zh97]. Especially, CHAFF [MM01] is a well-known example for this development. Current research investigates a better merge between the SAT and ATPG domain trying to exploit the best of the two worlds [GZ02].

4 Application: combinational equivalence checking

SAT and ATPG have proved to be important instruments in combinational equivalence checking. The task of combinational equivalence checking is to check whether or not two combinational circuits implement the same Boolean functions. Given two outputs y_A and y_B of two circuits A and B, it can be verified that y_A and y_B are equivalent by showing that $y_A \oplus y_B$ is unsatisfiable. For large circuits, however, this is generally intractable and even sophisticated solvers will fail in practice. Fortunately, the problem can be solved in many practical cases if the solving procedure is refined based on the following observation: most synthesis procedures perform many but fairly local circuit transformations. This preserves some of the original circuit structure so that the two designs of comparison contain many equivalent functions at their internal circuit nodes. These *internal equivalencies* [BT89] can be identified by passing from the circuit inputs to the outputs. A local analysis is usually sufficient to identify many internal equivalences. Previously determined equivalences serve as short cuts in the reasoning process so that more and more equivalences can be computed efficiently until the equivalence of the outputs is determined. The first equivalence checkers that showed the practicality of this paradigm were based on ATPG [Br93, Ku93]. Further work refined the process by also incorporating local BDDs [JM95, Ma96, KK97] and/or SAT [MG99, KG01] so that modern equivalence checkers can handle circuits with millions of gates.

5 Application: property checking

More recently checking properties of circuits became an area of intense research. It is regarded as one mean, some insiders even argue the only mean, to keep verification costs at an acceptable level. Another motivation is the increasing interest in reusing designs, or intellectual property (IP), in the context of system-on-chip (SoC). The business model for IP requires that the interface of an IP is specified as precisely as possible, for example with assertions. An interface contract contains assertions about what is required from the environment in which the IP is deployed. Then, the IP will provide certain properties. These properties are most naturally formulated as assertions. Clearly it is a business advantage to be able to formally prove that these assertions always hold.

Assertions come in two flavors: combinational and sequential. Checking combinational assertions can easily be formulated as a SAT problem by the Tsetin translation

discussed above. A typical example is checking for bus contention combinationally. With "combinationally" we mean that only the combinational logic is taken into account.

For example, if there are two potential drivers of a bus and their write enable signals are e_1 and e_2, respectively, assuming two valued signals only, then the SAT problem of checking for bus contention will use the CNF obtained by the Tsetin translation of the whole circuit and two additional sum terms consisting of a single literal each, e_1 and e_2, respectively. The addition of these two literals makes the CNF satisfiable if and only if both write signals can be asserted to one at the same time.

A weaker form of bus contention would only require that the write signals are never both asserted to one unless the driving values v_1 and v_2 are identical. To derive the required sum terms in this case may not look as easy as in the first case. But in general we can always encode such a property $(e_1 = e_2) \rightarrow (v_1 = v_2)$ as an assertion, which in turn can be translated into a circuit itself. The single output of this monitor is ´1´ if and only if the propery holds. Then we translate this monitor circuit into CNF as before. Finally it remains to add a single literal, which forces the output of the monitor to become ´0´ and check again for satisfiability. The CNF is satisfiable if and only if the property fails.

One could also check sequential designs for bus contention, where bus contention is only avoided for valid state assignments reachable after a proper reset. Similarly a property which says that a certain vector of signals is a one-hot encoding, can be checked combinationally, for instance if the signal vector is the output of combinational logic. In this case, we only check whether the vector is one-hot no matter in what state the system is. Sequentially checking the one-hot property means that the property only has two hold after a reset and we have to analyze the state space of the system.

Sequential property checking is also called *model checking* after [CE81, CG99]. In addition to checking assertions for all reachable states, model checking targets more involved properties, such as liveness. Liveness properties allow us to formulate the expected behaviour that necessarily will happen, such as a request will always be acknowledged. Even nested properties and relations between properties can be specified. Typically temporal logic is used for this purpose.

A standard approach for checking temporal properties is similar to the monitor circuit idea discussed above for combinational properties: the temporal formula is translated into a Büchi automata, a type of automata working on infinite execution sequences (traces), which is added to the circuit. The resulting system is checked for traces violating the temporal property.

The first algorithms to find such violating traces worked on the explicit state graph of the system, restricting their usage to circuits with a very small state space and a small set of primary inputs. With the invention of *symbolic model checking* [Mc93] it became possible to reason about much larger designs with more than 10^{20} states. The key idea was to use binary decision diagrams (BDDs) for the representation and manipulation of the characteristic functions for transition relations and sets of states.

Sequential property checking is much harder than combinational property checking. One can actually prove that it is PSPACE complete to check assertions for all reachable states of a design as opposed to NP completeness of combinational property checking. This is also reflected in the maximal size of the largest circuits that can typically checked. In practice model checking is restricted to designs with several hundreds of flip-flops while combinational property checking scales up to millions of gates.

If completeness is dropped and checking is only done to find bugs, similar to simulation, then sequential properties of much larger designs can be checked. This is the approach taken by *bounded model checking* (BMC) [BC99]. In addition it allows the use of SAT instead of BDDs which makes the checker much more robust. In BMC the sequential circuit is unrolled as in the time frame expansion approach for sequential ATPG. Then an additional monitor like formula is generated, restricting valid execution traces to be a counter example to the temporal formula being checked.

There are certain ideas to make BMC more complete, such as checking for diameters [BC99, BK02], using some type of restricted induction [SS00], or using SAT for image computation [Mc02]. In the future it may well happen that for certain applications BMC may replace BDD based model checking. Today BMC is already in widespread use as a fast filter before more costly temporal property checking algorithms based on BDDs are used. Finally it is apparent that some of the ideas that grew out of BMC can well be combined with sequential ATPG using ATPG as a replacement for SAT.

References

[AB90] M. Abramovici, M. Breuer, and A. Friedman, *Digital Systems Testing and Testable Design*. Computer Science Press, New York, 1990.

[AS01] F. A. Aloul and K. A. Sakallah, "SAT using ZBDDs," *in Dagstuhl Seminar 01051 on Computer Aided Design and Test -- BDDs versus SAT*, Schloss Dagstuhl, Germany, Jan/Feb. 01.

[BC99] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic Model Checking without BDDs", in *Proc. Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, April 1999.

[BS97] R. Bayardo Jr. and R. Schrag, "Using CSP look-back techniques to solve real-world SAT instances," in *Proc. Natl. Conf. on Artificial Intelligence*, pp. 203--208, 1997.

[BT89] C.L. Berman, and L.H. Trevillyan, "Functional Comparison of Logic Designs for VLSI Circuits", in *Proc. Intl. Conf. on Comp.-Aided Design* (ICCAD), pp. 456-459, 1989.

[Br93] D. Brand, "Verification of Large Synthesized Designs", *Proc. IEEE International Conference on Computer-Aided Design* (ICCAD), Santa Clara, pp. 534-537, Nov. 1993.

[BH84] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publishers, 1984.

[BK02] J. Baumgartner and A. Kühlmann, "Property Checking via Structural Analysis", *Proc. Intl. Conf on Computer Aided Verification*, July 2002.

[CA93] S. Chakradhar, V. D. Agrawal, and S. Rothweiler, "A transitive closure algorithm for test generation," *IEEE Trans. on CAD*, vol. 12, pp. 1015-1028, July 1993.

[CE81] E. Clarke, and E. Emerson. Synthesis of synchronization skeletons for branching time temporal logic, in *Proc. Logic of Programs Workshop*, 1981

[CG99] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*, MIT Press, 1999.

[DL62] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Communications ACM*, vol. 5, pp. 394--397, July 1962.

[DP60] M. Davis and H. Putnam, "A computing procedure for quantification theory," *Journal of the Association for Computing Machinery*, vol. 7, pp. 201-215, 1960.

[FS83] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms," *IEEE Trans. on Comp*., pp. 1137-1144, Dec. 1983.

[GZ02] M.K. Ganai, L. Zhang, P. Ashar, A. Gupta, and S. Malik, "Combining strengths of circuit-based and CNF-based algorithms for a high performance SAT-solver", in *Proc. Design Automation Conf. (DAC)*, pp. 747-750 , 2002.

[GB91] J. Giraldi and M. Bushnell, "Search state equivalence for redundancy identification and test generation," in *Proc. Intl. Test Conference*, pp. 184--193, 1991.

[GF01] E. Gizdarski and H. Fujiwara, "SPIRIT: A Highly Robust Combinational Test Generation Algorithm," 19th IEEE Proc. on VTS, 2001, pp 346-351.

[Go81] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Trans. on Comp*., vol. C-30, pp. 215-222, March 1981.

[JM95] Jain J., Mukherjee R., and Fujita M., "Advanced Verification Techniques Based on Learning", *Design Automation Conference* (DAC), pp. 420 - 426, June 1995.

[KM87] T. Kirkland and R. Mercer, "A topological search algorithm for ATPG," in *Proc. Design Automation Conference*, vol. 24, pp. 502--508, 1987.

[KB99] H. Kleine Büning and T. Lettmann. *Propositional logic: deduction and algorithms.* (Cambridge tracts in theoretical computer science 48) Cambridge University Press, 1999. ISBN-0-521-63017-7.

[KK97] A. Kühlmann, and F. Krohm, "Equivalence Checking using Cuts and Heaps", *Proc. Design Automation Conference*, San Fransisco, Juni 1997.

[KG01] A. Kühlmann, M. Ganai, V. Paruthi, "Circuit-based Boolean Reasoning", *Design Automation Conference* (DAC), pp. 232 - 237, June 2001.

[KP92] W. Kunz and D. Pradhan, "Recursive learning: An attractive alternative to the decision tree for test generation in digital circuits," in *Proc. Intl. Test Conference*, pp. 816-825, September 1992.

[Ku93] W. Kunz, "HANNIBAL: An Efficient Tool for Logic Verification Based on Recursive Learning", *Proc. Intl. Conference on Computer-Aided Design* (ICCAD), Santa Clara, pp. 538-543, Nov. 1993.

[Ha02] W. Kunz, J. Marques-Silva and S. Malik, "SAT and ATPG : Algorithms for Boolean Decision Making," in S. Hassoun and T. Sasao (Eds.) *Logic Synthesis and Verification*, Kluwer Academic Publishers, 2002. ISBN- 0-7923-7606-4.

[La89] T. Larrabee, "Efficient generation of test patterns using Boolean difference," in *Proc. Intl. Test Conference*, pp. 795-801, 1989.

[Ma86] R. Marlett, "An effective test generation system for sequential circuits," in *Proc. Design Automation Conf*., pp. 250-256, 1986.

[Mc93] K. McMillan, *Symbolic Model Checking : An Approach to the State Explosion Problem*, Kluwer Academic Publisher, 1993.

[Mc02] K. McMillan, "Applying SAT Methods in Unbounded Symbolic Model Checking", *Proc. Intl. Conf on Computer Aided Verification*, July 2002.

[MW85] S. Mallela and S. Wu, "A sequential circuit test generation system," in Proc. Intl. Test Conf., pp. 57-61, 1985.

[MG99] J. P. Marques-Silva and T. Glass, "Combinational equivalence checking using satisfiability and recursive learning," in *Proc. Design, Automation and Test in Europe Conf*., pp. 145-149, March 1999.

[MS99] J. P. Marques-Silva and K. A. Sakallah, "GRASP-A search algorithm for propositional satisfiability," *IEEE Trans. on Comp*., vol. 48, pp. 506-521, May 1999.

[Ma96] Y. Matsunaga, "An efficient equivalence checker for combinational circuits", *Proc. Design Automation Conference*, pp. 629-634, Las Vegas, 1996.

[MM01] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Engineering an efficient SAT solver," in *Proc. Design Automation Conf*., 2001., Part II (1970), pp. 115-125.

[RC90] J. Rajski and H. Cox, "A method to calculate necessary assignments in algorithmic test pattern generation," *in Proc. Intl. Test Conference*, pp. 25-34, 1990.

[Ro66] J. P. Roth, "Diagnosis of automata failures: A calculus and a method, "*IBM Journal of Research and Development*, vol. 10, pp. 278-291, July 1966.

[SA89] M. Schulz and E. Auth, "Improved deterministic test pattern generation with applications to redundancy identification," *IEEE Trans. on CAD*, vol. 8, pp. 811-816, July 1989.

[St89] G. Stålmarck, "A system for determining propositional logic theorems by applying values and rules to triplets that are generated from a formula," 1989, Swedish Patent 467 076, US Patent 5 276 897, European Patent 0 403454.

[SB96] P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, "Combinational test generation using satisfiability," *IEEE Trans. on CAD*, 1996.

[SS00] M. Sheeran, S. Singh, and G. Stålmarck, "Checking Safety Properties Using Induction and a SAT Solver," *in Proc. Intl. Conf. on Formal Methods in CAD*, November 2000.

[TG00] P. Tafertshofer, A. Ganz, and K. Antreich, "Igraine - an implication graph based engine for fast implication, justification, and propagation," *IEEE Trans. on CAD*, vol. 19, pp. 907-927, August 2000.

[Ts70] G.S. Tseitin, "On the Complexity of Derivation in Propositional Calculus", in A.O. Slikenko (Ed.*): Studies in Constructive Mathematics and Mathematical Logic*

[WR02] C. Wang, S. Reddy, I. Pomeranz, X. Lin, and J. Rajski, "Conflict Driven Techniques for Improving Deterministic Test Pattern Generation", in Proc. *IEEE International Conference on Computer-Aided Design* (ICCAD), Nov. 2002.

[Zh97] H. Zhang, "SATO: An efficient propositional prover," in *Proc. Intl. Conf. on Automated Deduction*, pp. 272-275, July 1997.