Using Interval Constraint Propagation for Pseudo-Boolean Constraint Solving

Karsten Scheibler and Bernd Becker
University of Freiburg, Georges Koehler Allee 51, 79110 Freiburg, Germany
{scheibler,becker}@informatik.uni-freiburg.de

***Abstract*—This work is motivated by (1) a practical application which automatically generates test patterns for integrated circuits and (2) the observation that off-the-shelf state-of-the-art pseudo-Boolean solvers have difficulties in solving instances with huge pseudo-Boolean constraints as created by our application.**

Derived from the SMT solver iSAT3 we present the solver iSAT3p that on the one hand allows the efficient handling of huge pseudo-Boolean constraints with several thousand summands and large integer coefficients. On the other hand, experimental results demonstrate that at the same time iSAT3p is competitive or even superior to other solvers on standard pseudo-Boolean benchmark families.

I. Introduction

Boolean satisfiability (SAT) and extensions thereof have gained increased importance also in the area of digital circuit testing – in particular since they allow the generation of so-called high quality tests [1], [2], [3]. On the other hand, it turns out that the test pattern generation for more complex physical defects demands for abilities going beyond the boolean level. In this paper we deal with pseudo-Boolean constraints (PB constraints) arising among others in this context and corresponding solution methods. Before going into solver details, we want to give a short introduction to our test pattern generation application. More details on the general context can be found e.g. in [4]. Some more specific information on the application considered here are provided in [5].

Assume the design of an integrated circuit (IC) is given and should now go into production. When manufacturing ICs, many things may go wrong. Thus, at the end of the production process it is necessary to test if the produced ICs behave according to their specification. I.e. one applies input patterns to an IC and compares the output with an expected result. If there is a difference, the IC is faulty. Obviously, the major aim is to recognize (hopefully all) faulty circuits from a given set of freshly produced ICs. Furthermore, the test procedure for one IC should be very fast in order to be able to test many circuits in a short period of time. Therefore, testing all possible input patterns is infeasible for circuits with a reasonable number of inputs.

To be able to generate a small set of test patterns, it is necessary to make assumptions about what can go wrong during the production process. Usually, the visible effects of a specific physical defect are described in a so-called fault model. Of course there exists a bunch of different fault models – each focussing on different aspects. The stuck-at fault model [6] assumes that a faulty line on a chip always carries a logical zero or one. Although it is one of the oldest models it is still widely-used, because of its simplicity. Nonetheless, due to to latest nanoelectronic technology [7], more complex fault models have become more and more important recently – in particular the open fault model. The open fault model looks at broken lines on a chip. Here, it is assumed that the voltage of the disconnected part is determined by the voltages of the surrounding lines. This voltage is then mapped to a logical value. In our application we focus on the generation of test patterns for this kind of fault.

When generating test patterns for a circuit one starts with a set of all possible faults regarding the underlying fault model. Then a fault is taken from this set and it is tried to generate a test pattern for it. Regarding the open fault model a set of boolean and PB constraints is created. The basic idea is to encode a fault-free and a faulty version of the circuit and demanding a difference at at least one output. In the faulty version additional PB constraints are used to describe the influence of the surrounding lines[1] (as given by the layout of the circuit) which induce faulty values to the disconnected part. If this set of constraints is satisfiable, the values of the variables representing the inputs of the circuit constitute a test pattern which discovers the considered fault.

One way to solve a set of PB constraints is to translate them into a SAT instance and to employ a SAT solver to solve it. The PB constraints generated within our application may contain up to several thousand summands. As our results show, such PB constraints pose a hard problem for solvers solely relying on SAT translation techniques. Therefore, we decided to utilize the SMT solver iSAT3, which is able to handle PB constraints directly in the solver. iSAT3 supports boolean, integer- and real-valued variables and uses interval constraint propagation (ICP) to handle boolean combinations of linear and non-linear constraints.

Compared to other solvers iSAT3 performs superior on our benchmark class. On the other hand one could expect that this is not the case for other benchmark classes as well, because ICP is a general deduction mechanism not tailored for PB constraints. In order to create a solver performing superior on all benchmark classes, we decided to develop a hybrid approach which (1) uses all the merits provided by SAT translation techniques and (2) exploits the abilities of ICP to do reasoning on the arithmetic level – in particular by introducing a preprocessing technique which is not applicable on the boolean level.

The paper is structured as follows. After giving some preliminaries in Section II, we present the extensions done to the solver in Section III. In Section IV we discuss the experimental results and conclude with a summary and outlook in Section V.

II. Preliminaries

Most modern SAT solvers operate on a *conjunctive normal form* (CNF). A CNF consists of a conjunction of clauses with each clause being a disjunction of literals and a literal being a boolean variable x or its negation $\overline{x}$. One core component of a SAT solver is the *boolean constraint propagation* (BCP) [8] which is used to detect implied assignments. Everytime a clause

[1] Each summand in the PB constraint (consisting of a large integer coefficient and a boolean variable) represents the logic value of a surrounding line (boolean variable) and its influence on the disconnected part (large integer coefficient).

ISBN: 978-0-9835678-4-4.

with n literals contains $n-1$ literals being already assigned to `false`, the remaining literal has to be `true` in order to retain a chance to satisfy the formula. Furthermore, today's SAT solvers add conflict clauses to the CNF to prune the search space even further – so-called *conflict-driven clause learning* (CDCL) [9].

In *SAT Modulo Theory* (SMT) the CDCL working principle is lifted to a higher level. The CNF is just a boolean abstraction of the real problem to be solved. Each literal may now represent a theory atom, e.g. $(x + y < 10)$. The SAT solver works on this boolean abstraction and assigns `true` or `false` to the literals – and thus also to the theory atoms. If the SAT solver does not find a solution the underlying SMT problem is unsatisfiable – but if it finds a satisfying assignment a theory solver has to be used to check if the conjunction of theory atoms satisfying the clauses is indeed satisfiable within the theory. If this is not the case, the boolean abstraction is refined with a conflict clause which forbids the conflicting theory atoms. This is the classical scheme for handling SMT formulas. It is also abbreviated as DPLL(T) or CDCL(T) – with T being the theory used within the atoms.

iSAT3 [10] is the third implementation of the iSAT algorithm [11], [12] and uses *interval constraint propagation* (ICP, see e.g. [13]) to check the consistency of the theory atoms. But unlike classical SMT, the iSAT algorithm does not separate the consistency check of the theory atoms from the search for a satisfying assignment in the boolean abstraction. Instead, ICP is tightly integrated into the CDCL framework. The iSAT algorithm allows theory atoms to contain linear and non-linear arithmetic as well as transcendental functions, e.g. $(x^2+y^2 = z^2)$, $(|v - w| < \min(v, w))$ or $(\sqrt[3]{x} + \sin y < e^z)$. Three variable types are natively supported: boolean, integer- and real-valued variables. Furthermore, ICP demands each integer- and real-valued variable to be declared with an initial interval.

iSAT3 uses an abstract syntax graph (ASG) to preprocess the given formula. In contrast to an abstract syntax tree (AST) an ASG-node may have multiple parent nodes. This allows structural hashing to natively share sub-expressions. The Tseitin-transformation [14] is used to convert the input formula to a CNF. Additionally, arithmetic constraints are decomposed into sub-expressions and simple bounds (a simple bound is a comparison between an integer- or real-valued variable and a constant). The solver core of iSAT3 is a SAT solver – extended in two directions: (1) it is able to create new literals on-the-fly during the solving process in order to map every newly deduced simple bound to a literal, (2) it executes ICP in addition to BCP. For more details refer to [10].

In the context of this paper we concentrate on constraints with pseudo-Boolean arithmetic. The linear form of such constraints has the form: $\sum_i c_i x_i \sim C$ where c_i and C are integer coefficients, x_i boolean variables and $\sim$ a relational operator with $\sim \in \{<, \leq, \geq, >\}$. Non-linear PB constraints additionally allow variables to be multiplied: $\sum_i (c_i \Pi_j x_j) \sim C$. The PB constraint $2x_1 + 4x_2 + x_3 < 5$ is an example for the linear form, while $3x_1x_4 + 3x_2 + x_3x_5 < 5$ represents a non-linear PB constraint.

Especially when translating PB constraints to SAT it is desired that the resulting CNF enables BCP to infer all the implications present in the original PB constraint – also denoted as *maintaining generalized arc consistency* (maintaining GAC). This means if a constraint C implies literal l under the partial assignment A then the constraint encoded in CNF C_{CNF} should allow BCP do the same: $C \wedge A \vDash l \;\Leftrightarrow\; C_{CNF} \wedge A \vdash_{BCP} l$.

In [15] BDDs, sorting networks and adder circuits were utilized to translate PB constraints into CNF. The proposed BDD-based encoding creates a BDD which describes the set of satisfying assignments of the PB constraint. Then each inner BDD-node is translated into CNF as an *if-then-else* (ITE) gate. While the BDD-based CNF encoding maintains GAC, sorting networks and adder circuits do not – this means possible implications are not recognized as early as possible which leads in most cases to a worse SAT solver performance. On the other hand the latter two encodings are compact, whereas BDD representations could have exponential size in worst case [16]. The authors of [17] proposed a different encoding which is also able to maintain GAC but stays polynomial in size. A PB constraint with n variables and the maximum integer coefficient c_{max} is encoded with $O(n^2 log(n) log(c_{max}))$ variables in $O(n^3 log(n) log(c_{max}))$ clauses. For PB constraints containing several hundreds or even thousands of variables this encoding method would generate billions of clauses and is therefore not applicable for PB constraints originating from our application. Additionally, BDDs are able to represent certain PB constraint types in linear size, while the encoding proposed in [17] stays in $O(n^3 log(n) log(c_{max}))$.

ICP operates on interval valuations and is used in iSAT3 to reason about linear and non-linear arithmetic constraints. Basically, ICP checks if a constraint is still consistent under the current (partial) assignment and tries to shrink the interval valuations of the variables occuring in the constraint if possible. In the following we illustrate the basic steps done by ICP when evaluating the PB constraint $C : 4x_1 + 2x_2 + 7x_3 < 10$ under the partial assignment $A : x_1 = 1$. With A these interval valuations are examined: $I_1 = 4x_1 =$ [4,4], $I_2 = 2x_2 =$ [0,2], $I_3 = 7x_3 =$ [0,7], $I_4 =$ [0,10). According to the current interval valuations C looks like this: [4,4] + [0,2] + [0,7] = [0,10). C is consistent under A, because there are still values in the intervals I_2 and I_3 such that the intersection between $I_1 + I_2 + I_3$ and I_4 is not empty. Furthermore, ICP is able deduce a new upper bound for I_3 (because of I_1). In order to prune definitive non-solutions I_3 is shrunk from [0,7] down to [0,6). In a next step the new upper bound for I_3 is propagated to x_3. With $I_3 = 7x_3 \;\wedge\; I_3 = [0,6) \;\wedge\; x_3 \in \mathbb{B}$ we can deduce $x_3 = 0$. The sum of the lower (upper) bounds of the left-hand side exceeds (falls below) the upper (lower) bound of the right-hand side, whenever the constraint is inconsistent under a partial assignment. Furthermore, the sum of the upper (lower) bound of interval I_i and the lower (upper) bounds of intervals $I_{j \neq i}$ exceeds (falls below) the upper (lower) bound of the right-hand side, whenever $x_i = 0$ $(x_i = 1)$. Therefore, ICP is able to maintain GAC. In fact this is not surprising, because ICP does reasoning on the arithmetic level. On the other hand ICP is a general deduction mechanism and not optimized for PB constraints. Especially PB constraints like $x_1 + x_2 + x_3 \geq 1$ are handled more efficently if their CNF translation is used – in this extreme case this would be just one clause: $(x_1 \vee x_2 \vee x_3)$. Therefore we combine ICP and BDD-based CNF translations as described in the next section.

III. iSAT3p = iSAT3 + PB Extensions

iSAT3p1: This variant is nearly identical to the underlying SMT solver iSAT3. We just extended the rewrite rules in the ASG formula preprocessing in order to normalize PB constraints to have positive coefficients on the left-hand side ($-c_i x_i \sim C$ can be rewritten to $c_i \overline{x_i} \sim C + c_i$ with $\sim \in \{<, \leq, \geq, >\}$).

iSAT3p2: We extend iSAT3p1 by adding the ability to represent PB constraints as BDDs similar to [15]. The boolean variables are ordered according to their coefficients – from the largest to the smallest. This also determines the static variable order of the BDD. The variable with the largest coefficient will be the top level variable. We use the ASG already present in iSAT3 to store the BDD as a directed acyclic graph of ITE-

ISBN: 978-0-9835678-4-4. Copyright owned jointly by the authors and FMCAD Inc.

nodes. These ITE-nodes are then converted to a CNF – which is handled efficiently by iSAT3p, because of its SAT solver core. A heuristic collects some statistics during BDD creation (i.e. number of created ITE-nodes, number of reused ITE-nodes because of structural hashing), estimates the expected size and decides whether the BDD creation should be aborted. If this is the case the PB constraint will be kept in its arithmetic form. The solver core will then use ICP as deduction mechanism.

iSAT3p3: On the one hand ICP is not as efficient as CNF translations for certain kinds of PB constraints. On the other hand ICP operates on the arithmetic level and therefore allows us to apply preprocessing techniques which are not applicable for CNF translations. We build on iSAT3p2 and extend it with symbolic gaussian elimination (SGE). The basic idea behind SGE is to generate helpful lemmas and add them to the formula before solving in order to strengthen ICP. The generated lemmas are not limited to PB constraints. In fact it does not matter, whether the variables occuring in a constraint are boolean, integer- or real-valued. We illustrate the idea with a small example with two constraints C_1, C_2 in the $\mathbb{R}^2$ space: $(y \geq 2.00001 \cdot x + 0.25) \wedge (y \leq 2 \cdot x)$, the initial intervals are: $x, y \in [0{,}1000000]$. Within the initial intervals C_1 and C_2 have no intersection. Therefore, the formula is unsatisfiable. ICP will continously shrink the intervals of x and y and may need millions of deductions until it finally discovers the conflict and deduces contradicting bounds for one variable. Geometrically, ICP constructs wrapping boxes around each constraint and calculates the intersection of those boxes. These boxes are parallel to the coordinate axes. Here, the idea is to generate an additional lemma which enables ICP to use an alternate coordinate axis for its wrapping box. A good choice for such an alternate axis is one of the constraints itself.

To generate such lemmas we re-use the auxiliary variables introduced during the decomposition of the original constraints into sub-expressions and simple bounds. Regarding our example the original constraints would be decomposed as follows.

$$\begin{array}{ll} C_1 \rightsquigarrow & (h_1 = y - 2.00001 \cdot x) \wedge (h_1 \geq 0.25) \\ C_2 \rightsquigarrow & (h_2 = y - 2 \cdot x) \wedge (h_2 \leq 0) \end{array}$$

Clearly, the following two equations are tautological and could be added to the formula without harm, because they just rephrase the equations above:

$$\begin{array}{l} y - 2.00001 \cdot x - h_1 = 0 \\ y - 2 \cdot x - h_2 = 0 \end{array}$$

In a system of equations, gaussian elimination replaces the problem variables step-by-step. We apply the same principle to the two tautolgies above. Assume we replace y in the second tautology with $2.00001 \cdot x + h_1$. This yields the following lemma: $0.00001 \cdot x + h_1 - h_2 = 0$. If we add it to the formula, ICP is able to deduce the conflict in a few steps: assume there is an additional auxiliary variable $(h_4 = -h_2)$ and we rewrite the lemma to $(0.00001 \cdot x + h_1 + h_4 = 0)$. Because of $(h_2 \leq 0)$ it directly follows that $(h_4 \geq 0)$. With $(h_1 \geq 0.25) \wedge (h_4 \geq 0)$ a new upper bound for x is deduced: $(x \leq -25000)$. This contradicts with the initial lower bound $(x \geq 0)$.

So in general SGE creates for every constraint a tautology containing the left-hand side of the constraint and the auxiliary variable introduced during Tseitin-transformation. Then, one of these tautologies is selected and redirected to a problem variable in order to replace this variable in all remaining tautologies. This process is repeated until no further replacements are possible. The current implementation processes the tautologies in the order of their creation in the ASG. Depending on the structure of the constraints this may result in one or more lemmas. On the one hand SGE needs enough constraints to construct useful lemmas, but on the other hand with increasing size and number of the constraints, SGE could become expensive. Therefore, a heuristic is used to decide if SGE should be aborted.

If the auxiliary variable representing the left-hand side of a constraint is used in a lemma, then this constraint will be kept – even if a BDD representation for this constraint is created later on. This allows ICP and BCP to reason about the same constraint simultaneously.

IV. Experimental Results

Solver		DBL (14)	DSL (355)	DSN (30)	OF10 (321)	$\sum$
Minisatp	SAT	8	136	-	8	
	UNS	**1**	93	-	0	
	S+U	9	229	-	8	243
SAT4JPB	SAT	9	129	[5]	221	
	UNS	0	90	**[5]**	**12**	
	S+U	9	219	[10]	233	461 [471]
Clasp	SAT	5	**138**	[8]	275	
	UNS	0	96	**[5]**	**12**	
	S+U	5	234	[13]	287	526 [539]
iSAT3p1	SAT	2	92	**[15]**	301	
	UNS	0	63	**[5]**	**12**	
	S+U	2	155	**[20]**	313	470 [490]
iSAT3p2	SAT	**13**	118	**[15]**	**307**	
	UNS	**1**	90	**[5]**	8	
	S+U	**14**	208	**[20]**	**315**	537 [557]
iSAT3p3	SAT	**13**	116	**[15]**	**307**	
	UNS	**1**	**122**	**[5]**	8	
	S+U	**14**	**238**	**[20]**	**315**	**567 [587]**
DEC-BIGINT-LIN=DBL, DEC-SMALLINT-LIN=DSL, DEC-SMALLINT-NLC=DSN, OPENFAULTS-DIV10=OF10						

Figure 1. Comparing Minisatp, Clasp and three variants of iSAT3p over a set of four benchmark families. The experiments were conducted on an Intel Xeon with 3.3 GHz with a timeout of 900 seconds and a memory limit of 8 GB.

We compared all three variants of iSAT3p against Minisatp [15] (git d91742bcd1), SAT4JPB [18] (version 2.3.5) and Clasp [19] (version 2.1.4). All three solvers were among the best solvers in the pseudo-Boolean competition 2012. Minisatp relies on the SAT solver Minisat (git 37dc6c67e2) and translates all PB constraints into SAT – either via BDD representations, sorting networks or adder circuits. SAT4JPB utilizes dedicated deduction mechanisms for PB constraints. Clasp is an answer set solver for (extended) normal logic programs.

From the pseudo-Boolean competition 2012 we selected those benchmark families containing satisfiability problems, namely: DEC-BIGINT-LIN (with 14 benchmark instances), DEC-SMALLINT-LIN (with 355 instances) and DEC-SMALLINT-NLC (with 30 instances). The first two families contain linear PB constraints, while the third contains non-linear ones. Additionally, we created a fourth benchmark family OPENFAULTS-DIV10 with 321 converted instances originating from our application. During test pattern generation we directly created the instance to be solved with ASG-nodes via the library interface of iSAT3p. In order to obtain a conjunction of PB constraints, we introduced additional auxiliary variables when needed. Furthermore, for PB constraints containing large numbers we had to divide all integer constants in the constraint by 10 – otherwise Clasp was unable to parse the benchmarks.

To compare the solvers we used an Intel Xeon with 3.3 GHz. The results are shown in Figure 1. For each benchmark family and for each solver the table shows the number of solved satisfiable (SAT) and unsatisfiable (UNS) instances as well as the sum of both (S+U). The best numbers in each category are

ISBN: 978-0-9835678-4-4.

marked bold. Minisatp did not handle benchmarks with non-linear PB constraints properly and immediately returned UNKNOWN for those benchmarks. Therefore, we list the numbers for the benchmark family DEC-SMALLINT-NLC for SAT4JPB, Clasp and iSAT3p in square brackets.

The results show that the baseline solver iSAT3p1 already outperforms Minisatp, SAT4JPB and Clasp on the benchmark families DEC-SMALLINT-NLC and OPENFAULTS-DIV10, but falls somewhat behind for DEC-BIGINT-LIN and DEC-SMALLINT-LIN. To a large extent this is due to the fact that the ICP routines borrowed from iSAT3 were written to handle generic linear and non-linear arithmetic constraints and are not optimized for PB constraints. iSAT3p2 is able to close the gap for DEC-BIGINT-LIN and DEC-SMALLINT-LIN. For these two benchmark families iSAT3p2 performs equally well as SAT4JPB with its dedicated PB deduction routines. Regarding OPENFAULTS-DIV10 and DEC-SMALLINT-NLC iSAT3p2 has significant lower runtimes compared to iSAT3p1. Finally, iSAT3p3 with SGE is able to outperform the other solvers on all benchmark families. As mentioned earlier, SGE needs on the one hand enough constraints to create useful lemmas, but on the other hand may become too expensive with increasing size and number of the constraints. Therefore, SGE is only applicable to a subset of the benchmark instances – in particular those in DEC-SMALLINT-LIN. The benchmark instances in DEC-BIGINT-LIN contain between 50-100 variables, but only two constraints. OPENFAULTS-DIV10 contains constraints with several hundred variables, so SGE will be too expensive and is aborted. DEC-SMALLINT-NLC contains non-linear PB constraints and is therefore not suitable for SGE.

The results for OPENFAULTS-DIV10 emphasize that solvers solely reying on a translation into SAT are not competitive for applications which require PB constraints with many summands and large integer coefficients. While Minisatp solves only 8 instances, SAT4JPB and Clasp solve 233 and 287 instances. All variants of iSAT3p solve almost all of the 321 instances.

To sum up: the results approve the efficacy of the extensions made to iSAT3. Resorting to BDD representations, the performance especially for the two benchmark families DEC-BIGINT-LIN and DEC-SMALLINT-LIN is improved. Here we see that a BDD-based CNF translation allows more efficient deductions. Additionally, SGE strengthens ICP and improves the overall performance further such that iSAT3p3 shows superior performance on all benchmark families.

V. Conclusion and Outlook

We presented an approach for solving PB constraints with interval constraint propagation – and when possible with BDD representations of the constraints. The experimental results confirmed the efficiency of our approach. Over the complete benchmark set iSAT3p3 was able to solve 587 instances – compared to the second best solver Clasp, this is a gain of 48 instances or 8.9%. The gain is even higher if iSAT3p3 is compared to SAT4JPB and Minisatp, namely 27.3% and 133% more benchmark instances are solved compared to these two solvers. Furthermore, we observed that methods only relying on a translation to SAT fail for our benchmark class. Therefore, it is clearly beneficial to keep the ability to handle constraints in an arithmetic way.

The fact that iSAT3p has a SAT solver in its core enables us to use all the merits of a BDD-based SAT translation. At the same time, the tight integration of ICP in iSAT3p allows us to opt out for BDD creation individually for each constraint. Additionally, we presented a preprocessing technique which generates lemmas to strengthen ICP reasoning. It improves the overall performance of the solver and is therefore a good starting point for future work in this direction. Furthermore, going beyond satisfiability checking and adding the capability to optimize solutions is a challenging task we want to address as well.

Acknowledgements

The authors thank Leonore Winterer and Felix Neubauer as well as Dominik Erb and Linus Feiten for supporting this work. This work has been partially supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (DFG, SFB/TR 14 AVACS, `http://www.avacs.org/`) and by the Cluster of Excellence BrainLinks-BrainTools (DFG, grant number EXC 1086)

References

[1] M. Sauer, A. Czutro, I. Polian, and B. Becker, "Small-delay-fault atpg with waveform accuracy," in *ICCAD*. IEEE, 2012, pp. 30–36.

[2] D. Erb, M. A. Kochte, M. Sauer, S. Hillebrecht, T. Schubert, H.-J. Wunderlich, and B. Becker, "Exact logic and fault simulation in presence of unknowns," *Accepted for publication in ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2014.

[3] S. Eggersglüß, R. Wille, and R. Drechsler, "Improved sat-based atpg: more constraints, better compaction," in *ICCAD*, J. Henkel, Ed. IEEE/ACM, 2013, pp. 85–90.

[4] N. K. Jha and S. K. Gupta, *Testing of Digital Systems*. Cambridge University Press, 2003.

[5] D. Erb, K. Scheibler, M. Sauer, and B. Becker, "Efficient smt-based atpg for interconnect open defects," in *DATE*, 2014, pp. 125:1–125:6.

[6] R. D. Eldred, "Test routines based on symbolic logical statements," *Journal of the ACM*, vol. 6, no. 1, pp. 33–36, 1959.

[7] V. H. Champac, R. Rodríguez-Montañés, J. A. Segura, J. Figueras, and J. A. Rubio, "Fault modelling of gate oxide short, floating gate and bridging failures in CMOS circuits," in *European Test Conf.*, 1991, pp. 143–148.

[8] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem proving," *Communications of the ACM*, vol. 5, pp. 394–397, 1962.

[9] J. P. M. Silva and K. A. Sakallah, "Grasp - a new search algorithm for satisfiability," in *ICCAD*, 1996, pp. 220–227.

[10] K. Scheibler, S. Kupferschmid, and B. Becker, "Recent improvements in the smt solver isat," in *MBMV*, C. Haubelt and D. Timmermann, Eds. Institut für Angewandte Mikroelektronik und Datentechnik, Fakultät für Informatik und Elektrotechnik, Universität Rostock, 2013, pp. 231–241.

[11] M. Fränzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert, "Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure," *Journal on Satisfiability, Boolean Modeling, and Computation*, vol. 1, no. 3-4, pp. 209–236, 2007.

[12] C. Herde, "Efficient solving of large arithmetic constraint systems with complex boolean structure: proof engines for the analysis of hybrid discrete-continuous systems," Ph.D. dissertation, 2011.

[13] F. Benhamou and L. Granvilliers, "Continuous and Interval Constraints," in *Handbook of Constraint Programming*, ser. Foundations of Artificial Intelligence, 2006, pp. 571–603.

[14] G. S. Tseitin, "On the complexity of derivations in propositional calculus," in *Studies in Constructive Mathematics and Mathematical Logics*, A. Slisenko, Ed., 1968.

[15] N. Eén and N. Sörensson, "Translating pseudo-boolean constraints into sat," *JSAT*, vol. 2, no. 1-4, pp. 1–26, 2006.

[16] I. Abío, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and V. Mayer-Eichberger, "A new look at bdds for pseudo-boolean constraints," *J. Artif. Int. Res.*, vol. 45, no. 1, pp. 443–480, Sep. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2444851.2444862

[17] O. Bailleux, Y. Boufkhad, and O. Roussel, "New encodings of pseudo-boolean constraints into cnf," in *SAT*, ser. Lecture Notes in Computer Science, O. Kullmann, Ed., vol. 5584. Springer, 2009, pp. 181–194.

[18] D. L. Berre and A. Parrain, "The sat4j library, release 2.2," *JSAT*, vol. 7, no. 2-3, pp. 59–6, 2010.

[19] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, "*clasp* : A conflict-driven answer set solver," in *LPNMR*, ser. Lecture Notes in Computer Science, C. Baral, G. Brewka, and J. S. Schlipf, Eds., vol. 4483. Springer, 2007, pp. 260–265.