Accurate Computation of Sensitizable Paths using Answer Set Programming*

Benjamin Andres[1], Matthias Sauer[2], Martin Gebser[1], Tobias Schubert[2], Bernd Becker[2], and Torsten Schaub[1]

[1] University of Potsdam
August-Bebel-Strasse 89
14482 Potsdam, Germany
{ bandres | gebser | torsten }@cs.uni-potsdam.de
[2] Albert-Ludwigs-University Freiburg
Georges-Köhler-Allee 051
79110 Freiburg, Germany
{ sauerm | schubert | becker }@informatik.uni-freiburg.de

Abstract. Precise knowledge of the longest sensitizable paths in a circuit is crucial for various tasks in computer-aided design, including timing analysis, performance optimization, delay testing, and speed binning. As delays in today's nanoscale technologies are increasingly affected by statistical parameter variations, there is significant interest in obtaining sets of paths that are within a length range. For instance, such path sets can be used in the emerging areas of *Post-silicon validation and characterization* and *Adaptive Test*. We present an ASP-based method for computing well-defined sets of sensitizable paths within a length range. Unlike previous approaches, the method is accurate and does not rely on a priori relaxations. Experimental results demonstrate the applicability and scalability of our method.

1 Introduction

Precise knowledge of the longest sensitizable paths in a circuit is crucial for various tasks in computer-aided design, including timing analysis, performance optimization, delay testing, and speed binning. However, the delays of individual gates in today's nanoscale technologies are increasingly affected by statistical parameter variations [1]. As a consequence, the longest paths in a circuit depend on the random distribution of circuit features [12] and are thus subject to change in different circuit instances. For this reason, there is significant interest in obtaining sets of paths that are within a length range, in contrast to only the longest nominal path as in classical small delay testing [13]. Among other applications, such path sets can be used in the emerging areas of *Post-silicon validation and characterization* [8] and *Adaptive Test* [14].

Comprehensive test suites are generated and used in the circuit characterization or yield-ramp-up phase. The inputs to be employed in actual volume manufacturing test are chosen based on their observed effectiveness in detecting defects, In general the

* This work was published as a poster paper in [3].

quality of a delay test tends to increase with the delay of the actually tested path. However, a pair t_1 of test inputs may be more effective than a pair t_2, even though t_1 sensitizes a shorter path than t_2. Modeling inaccuracy is one of the reasons leading to such mismatches. For instance, while the sum of gate delays along the path sensitized by t_1 may be smaller than the sum for t_2, the pair t_1 could induce crosstalk or IR-drop, increasing the signal propagation delay along the path. These effects are generally difficult to model during timing analysis, and also affected by process variations. High-quality Automatic Test Pattern Generation (ATPG) methods should be able to control the path length and generate a large number of alternative test pairs that sensitize different paths of predefined length, to be applicable for adaptive test.

While structural paths can be easily extracted from a circuit architecture, many structural paths are not sensitizable and therefore present *false paths* [7]. The usage of such false paths leads to overly pessimistic and inaccurate results. Therefore, determination of path sensitization is required for high-quality results, although it constitutes a challenging task that requires complex path propagation and sensitization rules.

In order to reduce the algorithmic overhead, various methods for the computation of sensitizable paths make use of relaxations [11], making trade-offs between complexity and accuracy. Methods based on the sensitization of structural paths [15, 6] restrict the number of paths they consider for accelerating the computation and to limit memory usage. Due to these restrictions, however, they may miss long paths. Recent methods [17, 16] based on Boolean Satisfiability (SAT; [5]) have shown good performance results but are limited in the precision of the encoded delay values. As their scaling critically depends on delay resolution, such methods are hardly applicable when high accuracy is required.

We present an exact method for obtaining longest sensitizable paths, using Answer Set Programming (ASP; [4]) to encode the problem. ASP has become a popular approach to declarative problem solving in the field of Knowledge Representation and Reasoning (KRR). Unlike SAT, ASP provides a rich modeling language as well as a stringent semantics, which allows for succinct representations of encodings.

In what follows, we assume some familiarity with ASP, its semantics as well as its basic language constructs. A comprehensive treatment of ASP can be found in [4, 10]. Our encodings are written in the input language of *gringo* 3 [9].

The remainder of the paper is structured as follows. Section 2 provides our ASP encoding of sensitizable paths. Experimental results are presented in Section 3. Section 4 concludes the paper.

2 ASP Encoding

The general setting of longest sensitizable path calculation for a Boolean circuit and a *test gate* g is displayed in Figure 1. Observe that gates in the *input cone* A_1 influence g, while those in the *output cone* A_2 depend on g. Furthermore, the additional gates in A_3 have an impact on gates in A_2. Then, a (longest) sensitizable path including the test gate g is determined by *two* truth assignments (modelling two time frames) to the primary input gates in $A_1 \cup A_3$ such that the truth values of g and one output gate in A_2 get flipped over the two time frames.

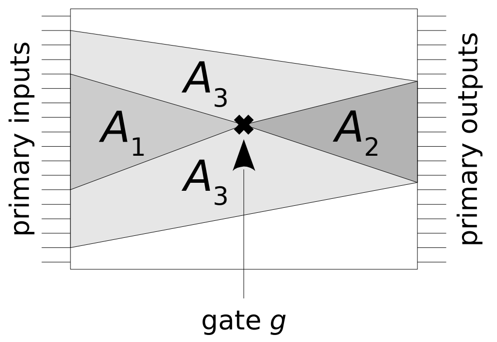

Fig. 1. Input/Output cones for longest sensitizable path calculation.

As common in ASP, we represent the problem of calculating a longest sensitizable path by facts describing a problem instance along with a generic encoding. First, a Boolean circuit and gate delays are described by facts of the following form:

`in(g).`	for each primary input gate `g`.
`nand(g).`	for each non-input gate `g`.
`out(g).`	for each output gate `g`.
`test(g).`	for the test gate `g`.
`wire(g1,g2,r,f).`	for a connection from `g1` to `g2`.

Facts of the first three forms provide constants `g` standing for input, non-input, and output gates, respectively, of a circuit, where we assume either `in(g)` or `nand(g)` to hold for each gate `g`. A fact of the fourth form specifies the test gate `g` in the circuit. Finally, the inputs `g1` of a gate `g2` are given by facts of the fifth form, where the integers `r` and `f` provide the delays of a `r`ising or `f`alling edge at `g2`, respectively. W.l.o.g., we here limit the attention to NAND non-input gates; further Boolean functions could be handled by extending the encoding in Figure 2. We also assume that output gates do not serve as inputs to other gates.

Our generic encoding of longest sensitizable paths is shown in Figure 2. Given that gate delays are only required for path length maximization, but not for the actual path calculation, the rule in Line 1 projects instances of `wire(G1,G2,R,F)` (given by facts) to connected gates `G1` and `G2`. Then, starting from the test gate in the circuit, the rules in Line 3 to 6 inductively determine all gates of the input and output cone, respectively (cf. A_1 and A_2 in Figure 1). The union of the input and output cone is represented by the instances of `inocone(G)` derived by the rules in Line 8 and 9. In Line 10 and 11, such instances are taken as starting points to inductively determine the set of all relevant gates ($A_1 \cup A_2 \cup A_3$ in Figure 1), given by the derived instances of `allcone(G)`. Note that the rules in Line 1–11 are deterministic, yielding a unique least model relative to facts.

The calculation of a path through the test gate `g` is implemented by the rules in Line 13 and 14. It starts in Line 13 by choosing exactly one output gate from the output cone, represented by an instance of `path(G)`. In Line 14, the path is continued backwards including exactly one predecessor gate for every non-input gate already on

```
wire(G1,G2) :- wire(G1,G2,R,F).

inpcone(G2) :- test(G2).
inpcone(G1) :- inpcone(G2), wire(G1,G2).
outcone(G1) :- test(G1).
outcone(G2) :- outcone(G1), wire(G1,G2).

inocone(G2) :- inpcone(G2).
inocone(G2) :- outcone(G2).
allcone(G2) :- inocone(G2).
allcone(G1) :- allcone(G2), wire(G1,G2).

1 { path(G2) : outcone(G2) : out(G2) } 1.
1 { path(G1) : inocone(G1) : wire(G1,G2) } 1 :- path(G2), not in(G2).

{ one(G1) } :- allcone(G1), in(G1).
one(G2) :- allcone(G2), nand(G2), wire(G1,G2), not one(G1).
{ two(G1) } :- allcone(G1), in(G1).
two(G2) :- allcone(G2), nand(G2), wire(G1,G2), not two(G1).

flipped(G) :- inocone(G), one(G), not two(G).
flipped(G) :- inocone(G), two(G), not one(G).
:- path(G), not flipped(G).

delay(G2,M) :- path(G1), path(G2), wire(G1,G2),
M = #min[ wire(G1,G2,R,F) = R, wire(G1,G2,R,F) = F ].
add(G2,R-F) :- path(G1), path(G2), wire(G1,G2,R,F), R > F, two(G2).
add(G2,F-R) :- path(G1), path(G2), wire(G1,G2,R,F), R < F, one(G2).

#maximize[ delay(G2,M) = M, add(G2,N) = N ].
```

Fig. 2. ASP encoding for calculating longest sensitizable paths in a circuit.

the path. Since any path from gates in the input cone to those in the output cone must include `g`, the restriction of path elements to their union (instances of `inocone(G)`) makes sure that `path(g)` holds. Also note that, although path calculation is logically encoded backwards, ASP solving engines are not obliged to proceed in any such order upon searching for answer sets.

The truth assignments needed for checking whether a path at hand is sensitizable are generated by the rules in Line 16 to 19. To this end, for each relevant input gate `g1` of the circuit (`allcone(g1)` and `in(g1)` hold), choice rules allow for guessing *two* truth values. In fact, the atoms `one(g1)` and `two(g1)` express whether `g1` is true in the first and the second time frame, respectively. Given the values guessed for input gates, NAND gates `g2` are evaluated accordingly, and the outcomes are likewise represented by `one(g2)` and `two(g2)`. For gates `g` in the input or output cone, which can possibly belong to a calculated path, the rules in Line 21 and 22 check whether their truth values are sensitizable; if so, it is indicated by deriving `flipped(g)`. Finally, the integrity constraint in Line 23 stipulates that each gate on the calculated path must be flipped, thus denying truth assignments whose transition does not propagate along the whole path.

In order to calculate the longest sensitizable paths, the rule in Line 25 derives a delay incurred whenever two gates `g1` and `g2` are connected along a path. This delay, given by the minimum of `r` and `f` in `wire(g1,g2,r,f)` (specified by a fact), can be obtained conveniently via *gringo*'s `#min` aggregate [9]. Furthermore, if `r` and `f` diverge, an additional delay `r`$-$`f` is incurred in case that `r` $>$ `f` and `g2` is flipped to true (Line 26), or `f`$-$`r` when `g2` is flipped to false and `r` $<$ `f` (Line 27). Note that considering only `one(g2)` or `two(g2)`, respectively, is sufficient here because the integrity constraint in Line 23 checks that the truth value of `g2` is indeed flipped. While (additional) delays derived via the rules in Line 26 and 27 depend on a path and truth assignments, the basic delay in Line 25 is obtained as soon as connected gates `g1` and `g2` are on a path. Since it does not consider truth assignments, the rule in Line 25 relies on fewer vagrant prerequisites and is thus "easier to apply" upon searching for answer sets. The main objective of calculating longest sensitizable paths is expressed by the `#maximize` statement in Line 29, which instructs ASP solving engines to compute answer sets such that the sum of associated gate delays is as large as possible.

3 Experimental Results

We evaluate our method on ISCAS85 and the combinatorial cores of ISCAS89 benchmark circuits, given as gate-level net lists. Path lengths are based on a pin-to-pin delay model with support for different rising-falling delays. The individual delay values have been derived from the Nangate 45nm Open Cell Library [2]. Below, we report sequential runtimes of the ASP solver *clasp* (version 2.0.4) on a Linux machine equipped with 3.07GHz Intel i7 CPUs and 16GB RAM.

Figure 3 shows the workflow for testing a circuit. At the start, the ASP instance describing the circuit and our generic encoding (cf. Figure 2) are grounded by *gringo*. Different from the modeling in Section 2, here, we do not specify a test gate within the ASP instance for *g*, but rather add a corresponding fact after grounding. To obtain a

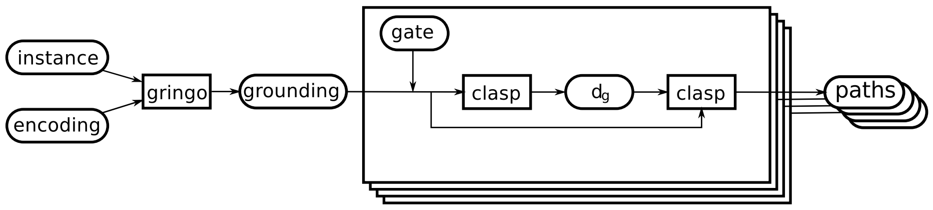

Fig. 3. Workflow of the experiments.

grounding amenable to arbitrary test gates, instead of facts, we used choice rules for a priori leaving a gate to test open. Given that sensitizable paths are computed in a loop over all gates to be analyzed, the reuse of the same grounding saves some overhead by not rerunning *gringo* for each test gate. However, note that such preprocessing "optimization" has no influence on the runtimes of *clasp* and thus does not affect solving time measurements. The grounding augmented with a test gate g serves as input for *clasp*, which in its first run performs optimization to identify a longest sensitizable path with maximum delay d_g. With d_g at hand, we further proceed to computing all paths with a delay equal or greater than $r = 0.95 * d_g$. This is accomplished by reinvoking *clasp* with the command-line parameters `--opt-all=`r and `--project` to enumerate all sensitizable paths within the range $[r, d_g]$. While the first parameter informs *clasp* about the quality threshold r for sensitizable paths to enumerate, the second is used to omit repetitions of the same path with different truth assignments. As a consequence, *clasp* enumerates distinct sensitizable paths, whose delay is at least r, without repetitions. An overlaying python program reuses the information of d_g and paths found in previous iterations to decide whether subsequent gates need to be analysed and ensures that *clasp* does not need to calculate the same paths for different gates.

Table 1 displays the runtimes of our method using a length-preserving mapping (avoiding rounding errors) of real-valued gate delays to integers. "Circuit" and "Gates" indicate a particular benchmark circuit along with its number of gates to be tested. The next three columns give statistics for the search for longest sensitizable paths, displaying the average runtime per solver call, the sum of runtimes for all gates in seconds and the number of solver calls needed to calculate d_g for all gates. The three columns below "Path set" provide statistics for the enumeration of distinct sensitizable paths with length at least r. Here, we show the average runtime for enumerating 1000 paths, the sum of runtimes for all gates, and finally the total number of different paths found. The columns below "Total" summarize both computation phases of *clasp*, optimization and enumeration. The first column present the total number *clasp* was called. Finally, the last two columns provide the total solving time of *clasp* for both computation passes and the total runtime needed for the benchmark. Please note, that the smallest resolution for measuring the solving time of *clasp* is $0.01s$. Thus, solving time results for circuits with less than $0.01s$ per gate may be inacurate up to the number of solver calls times $0.01s$.

As can be seen in Table 1, the scaling of our method is primarily dominated by the number of gates in circuits. Over all circuits, the average runtime for processing one

Circuit	Gates	Longest path (d_g)			Path set (95%)			Total		
		Time in s per call	Time in s	Calls	Time in s per 1000 paths	Time in s	Paths	Solver calls	Solving Time in s	Total time in s
c0017	6	< 0.01	< 0.01	3	< 0.01	< 0.01	8	7	< 0.01	0.05
c0095	27	< 0.01	< 0.01	7	< 0.01	< 0.01	90	22	< 0.01	0.23
c0432	160	0.05	2.46	53	0.24	4.67	19356	112	7.13	11.67
c0499	202	0.01	0.64	64	0.49	0.94	1928	160	1.58	5.54
c0880	383	< 0.01	0.33	82	0.21	0.77	3617	212	1.10	7.78
c1355	546	0.29	18.87	64	1.36	32.54	23936	160	51.41	63.60
c1908	880	0.25	34.37	137	2.00	64.33	32174	378	98.70	131.22
c2670	1269	0.01	5.30	440	1.41	8.05	5700	1023	13.35	101.79
c3540	1669	2.26	544.32	241	10.42	1125.60	107994	697	1669.92	1799.69
c5315	2307	0.05	25.43	485	2.02	39.65	19603	1206	65.08	266.83
c7552	3513	0.04	24.59	576	1.97	40.77	20745	1622	65.36	444.07
cs00027	10	< 0.01	< 0.01	3	< 0.01	< 0.01	11	7	< 0.01	0.03
cs00208	104	< 0.01	< 0.01	33	< 0.01	< 0.01	97	98	< 0.01	0.62
cs00298	119	< 0.01	< 0.01	48	< 0.01	< 0.01	137	126	< 0.01	1.09
cs00344	160	< 0.01	< 0.01	45	< 0.01	< 0.01	169	125	< 0.01	1.14
cs00349	161	< 0.01	< 0.01	47	< 0.01	< 0.01	170	127	< 0.01	1.93
cs00382	158	< 0.01	< 0.01	48	< 0.01	< 0.01	169	144	< 0.01	2.03
cs00386	159	< 0.01	< 0.01	30	< 0.01	< 0.01	124	142	< 0.01	2.08
cs00400	162	< 0.01	< 0.01	49	< 0.01	< 0.01	184	149	< 0.01	2.15
cs00420	218	< 0.01	0.01	66	< 0.01	< 0.01	305	205	0.01	3.19
cs00444	181	< 0.01	< 0.01	49	< 0.01	< 0.01	210	162	< 0.01	2.45
cs00510	211	< 0.01	< 0.01	58	< 0.01	< 0.01	230	161	< 0.01	2.06
cs00526	194	< 0.01	< 0.01	74	< 0.01	< 0.01	247	190	< 0.01	2.17
cs00641	379	< 0.01	0.01	68	0.06	0.02	326	236	0.03	5.74
cs00713	393	< 0.01	0.01	84	0.06	0.02	309	276	0.03	5.20
cs00820	289	< 0.01	0.02	71	< 0.01	< 0.01	361	273	0.02	5.36
cs00832	287	< 0.01	0.02	73	< 0.01	< 0.01	372	269	0.02	5.02
cs00838	446	< 0.01	0.05	140	0.18	0.15	853	444	0.20	12.31
cs00953	418	< 0.01	0.01	111	0.03	0.01	342	354	0.02	7.48
cs01196	530	< 0.01	0.39	145	0.62	0.34	550	417	0.73	14.90
cs01238	509	< 0.01	0.54	144	0.72	0.42	586	400	0.96	13.39
cs01423	657	< 0.01	1.51	184	0.87	1.94	2236	529	3.45	25.05
cs01488	653	< 0.01	0.02	155	0.02	0.01	517	676	0.03	22.10
cs01494	647	< 0.01	0.02	157	0.02	0.01	521	654	0.03	21.69
cs05378	2779	< 0.01	0.65	506	0.32	1.71	5334	1759	2.36	320.37
cs09234	5597	< 0.01	6.82	795	0.69	13.54	19703	3483	20.36	1091.54
cs13207	8027	0.02	27.15	1332	2.97	53.41	18011	5864	80.56	2833.38
cs15850	9786	0.66	973.07	1480	9.56	3172.45	331964	6322	4145.52	9825.64
cs35932	16353	< 0.01	0.06	5321	0.41	5.9	14321	13463	5.96	16437.94
cs38584	19407	< 0.01	33.23	7266	2.35	65.18	27722	20227	98.41	42700.61

Table 1. Application using exact delay values

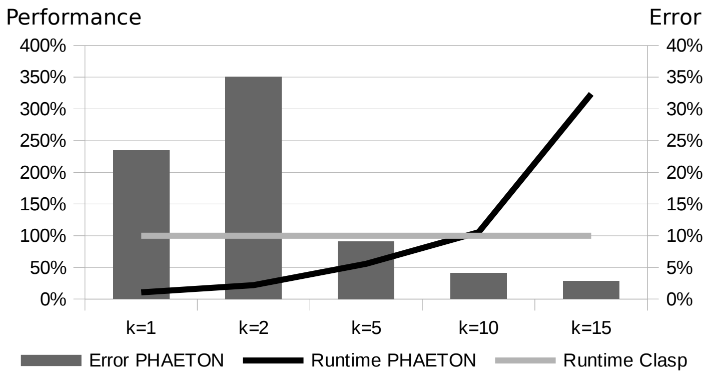

Fig. 4. Comparison with PHAETON [17] using ISCAS85 circuits

test gate is rather low and often within fractions of a second. In addition, our method allows for enumerating the complete set of sensitizable paths within a given range in a single solver call, thus avoiding any expenses due to rerunning our solver. This allows us to enumerate thousands of sensitizable paths and test pattern pairs sensitizing them very efficiently. In fact, the overhead of path set computation compared to optimization in the first phase is relatively small, even for complex circuits. E.g., for the c3540 circuit, 2.26 seconds are on average required for optimization, and 10.42 seconds on average per 1000 enumerated paths. The rather large discrepancy between solving and total runtime for large, computational easy circuits, e.g. cs13207, is explained by the fact that *clasp* currently needs to read the grounded file from the disc for every call. To overcome this bottleneck we hope to utilize *iclingo*, an incremental ASP system implemented on top of *gringo* and *clasp*, in future work as soon as *iclingo* supports `#maximize` statements. This would allow us to analyze all gates of a circuit within a single solver call, thus drastically reducing the disc access. In addition, the *iclingo* could reuse information gained from previously processed gates for solving successive gates, efficiently.

In order to demonstrate the scaling of our approach wrt delay accuracy, we also used different mappings of real-valued delays to integers, and corresponding runtime results for the ISCAS85 benchmark set as shown in Table 2. In addition to the exact mode used in the previous experiment, we employed a rounding method to five delay values, shown in the columns labeled with “5”. Likewise, we applied rounding to 1000 delay values. As before, we report average runtimes per call in seconds for the two phases of optimizing sensitizable path length and of performing enumeration. Considering the results, we observe that runtimes of *clasp* are almost uninfluenced by the precision of gate delays. This is explained by the fact that weights used in `#minimize` or `#maximize` statements do influence the space of answer sets wrt to which optimization and enumeration are applied. In the ISCAS89 benchmark set the solving time per call was almost universally less than $0.01s$.

We compared our method with an SAT-based approach called “PHAETON“ proposed in [17]. The results are shown in Figure 4. The Figure shows the runtime needed by PHAETON to compute 1000 paths for ISCAS85 benchmark circuits with different levels of accuracy indicated by the number of delay steps k. In order to compare the results of the proposed method with PHAETON, the runtime is given as percent on the primary x-axis, with 100% being our method. The secondary x-axis gives the discretization error of PHAETON. As can be seen, for low accuracy levels which result in an average discretization error of around 5%, PHAETON scales better than our optimal approach. However, for increased accuracy levels, the proposed method outperforms PHAETON and is therefore better suited for precise computation of longest sensitizable paths.

Circuit	Time (d_g) per call			Time (95%) per call		
	5	1000	exact	5	1000	exact
c0017	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
c0095	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
c0432	0.04	0.05	0.05	0.07	0.08	0.08
c0499	0.01	0.01	0.01	0.01	0.01	0.01
c0880	< 0.01	0.01	< 0.01	0.01	0.01	0.01
c1355	0.13	0.22	0.29	0.15	0.21	0.34
c1908	0.20	0.31	0.25	0.20	0.47	0.27
c2670	0.01	0.01	0.01	0.01	0.02	0.01
c3540	2.24	2.55	2.26	2.50	2.63	2.47
c5315	0.04	0.06	0.05	0.04	0.08	0.05
c7552	0.04	0.05	0.04	0.04	0.07	0.04

Table 2. Delay accuracy comparison

4 Conclusions

We presented a method for the accurate computation of sensitizable paths based on a flexible and compact encoding in ASP. Unlike previous methods, our approach does not rely on a priori relaxations and is therefore exact. We demonstrated the applicability and scalability of our method by extensive experiments on ISCAS85 and ISCAS89 benchmark circuits.

Future work includes further efforts to optimize the ASP encoding by incorporating additional rules, with the goal of reducing the search space and helping *clasp* to discard unsatisfactory sensitizable paths faster. Another way to improve runtime is to specialize *clasp*’s search strategy to the problem of calculating (longest) sensitizable paths.

Acknowledgments. Parts of this work are supported by the German Research Foundation under grant GRK 1103, SCHA 550/8-3 and SCHA 550/9-1.

References

1. International Technology Roadmap For Semiconductors. Available at `http://www.itrs.net`.
2. Nangate 45nm open cell library. Available at `http://www.nangate.com`.
3. B. Andres, M. Sauer, M. Gebser, T. Schubert, B. Becker, and T. Schaub. Accurate computation of longest sensitizable paths using answer set programming. In R. Drechsler and G. Fey, editors, *Sechste GMM/GI/ITG-Fachtagung für Zuverlässigkeit und Entwurf (ZuE'12)*, 2012.
4. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
5. A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.
6. Jaeyong Chung, Jinjun Xiong, V. Zolotov, and J. Abraham. Testability driven statistical path selection. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 417 –422, june 2011.
7. Olivier Coudert. An efficient algorithm to verify generalized false paths. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 188 –193, june 2010.
8. Prasanjeet Das and Sandeep K. Gupta. On generating vectors for accurate post-silicon delay characterization. In *Test Symposium (ATS), 2011 20th Asian*, pages 251 –260, nov. 2011.
9. M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo`. Available at `http://potassco.sourceforge.net`.
10. M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012.
11. Jie Jiang, Matthias Sauer, Alexander Czutro, Bernd Becker, and Ilia Polian. On the optimality of k longest path generation algorithm under memory constraints. In *Design, Automation and Test in Europe (DATE)*, 2012.
12. K. Killpack, C. Kashyap, and E. Chiprout. Silicon speedpath measurement and feedback into eda flows. In *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, pages 390 –395, june 2007.
13. M.M.V. Kumar and S. Tragoudas. High-quality transition fault ATPG for small delay defects. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(5):983 –989, may 2007.
14. P. Maxwell. Adaptive test directions. In *Test Symposium (ETS), 2010 15th IEEE European*, pages 12 –16, may 2010.
15. Wangqi Qiu and D.M.H. Walker. An efficient algorithm for finding the k longest testable paths through each gate in a combinational circuit. In *Test Conference, 2003. Proceedings. ITC 2003. International*, volume 1, pages 592 – 601, 30-oct. 2, 2003.
16. M. Sauer, A. Czutro, T. Schubert, S. Hillebrecht, I. Polian, and B. Becker. SAT-based analysis of sensitisable paths. In *Design and Diagnostics of Electronic Circuits Systems (DDECS), 2011 IEEE 14th International Symposium on*, pages 93 –98, april 2011.
17. Matthias Sauer, Jie Jiang, Alexander Czutro, Ilia Polian, and Bernd Becker. Efficient SAT-based search for longest sensitisable paths. In *Test Symposium (ATS), 2011 20th Asian*, pages 108 –113, nov. 2011.