Performance analysis of constraint solvers for Coverage Directed Test Generation

Jomu George Mani Paret and Otmane Ait Mohamed

Department of Electrical and Computer Engineering,
Concordia University, Montreal, Canada
{jo_pare,ait}@ece.concordia.ca

Technical Report

January, 2012

Abstract

Coverage is a metric used to obtain information about execution of hardware description language (HDL) statements. It helps to determine how well the test cases verified the design under verification. Coverage directed test generation (CDTG) techniques analyze coverage results and adapt the test generation process to improve the coverage. This step is iterated until the required coverage is attained. One of the important components of CDTG technique is the constraint solver. The efficiency of CDTG depends on the constraint solver used. In this report, a comparative study is conducted to measure the effectiveness of VCS2009.06 against other commercially available constraint solvers. Our results are obtained by solving N Queens problem and car sequencing problem using the different solvers.

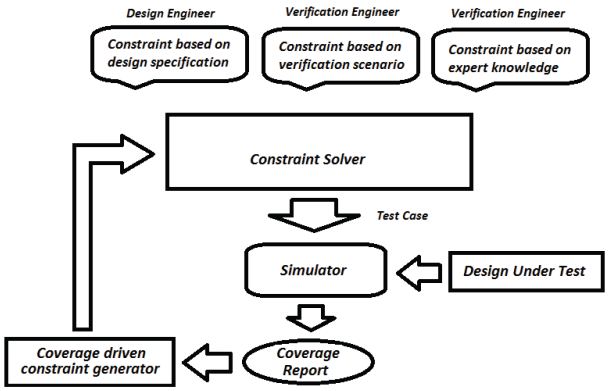

Fig. 1: Coverage Driven Constraint Random Test Generator

1 Introduction

Simulation based verification (SV) is the most commonly used method for the verification of large designs. SV tries to find errors in a design by using a directed or pseudo-random simulation tests. Although SV can be very effective, its success depends heavily on the quality of the tests in use. The number of test cases valid for a particular design is limited. All the valid test cases are not of interest since the verification engineers are concentrating on certain scenarios. In constraint random test (CRT) generation method the conditions for valid test cases and conditions for the scenarios are specified. Solving the constraints will give the required test case or cases. Hence out of the many test generation methods that have been developed, constraint random test (CRT) generation is the most commonly used for the verification of complex design. In CRT constraints are manually specified in order to hit areas or specific scenarios in the design that are not covered. In order to find out whether all the scenarios and corner cases are covered, coverage analysis has to be done. Coverage data gives the necessary information regarding the completeness of the verification process. Generating test cases to attain 100% coverage is a key challenge. Coverage Directed Test Generation (CDTG) is a methodology which uses coverage data to direct the next round of test generation towards producing tests that increase coverage percentage. This takes large amount of engineering skill and is time-consuming. It is also an error prone process and hence automation of this process is beneficial. This is achieved by studying existing tests and the coverage percentage. Automated coverage directed constraint random test generation is a technique to automate the feedback from coverage analysis to test generation [1, 2, 3]. A large amount of work has been done in CDTG to find the best way to automate the process of effective constraint generation. Nowadays data mining techniques and neural networks are used for attaining 100% coverage [4, 5].

Although there are different features in different technologies developed by different group independently, all of them agree on one point. The CDTG must have two parts:

- Constraint models or language used to describe the constraints.
- Constraint solver engine used to find the solution or solutions for the given constraints.

Research is going on to develop effective constraint solver for CDTG [6]. The test cases are generated in CDTG by solving the constraints and then the results are used to generate more constraints which will help to attain maximum coverage. Hence the test generation methods of CDTGs are equivalent to a constraint satisfaction problem (CSP). Therefore the efficiency

of a CDTG is dependent on the constraint solver used. In this paper, we try to compare VCS 2009.06 with two different constraint solvers (engine) which are based on different constraint models or languages and a constraint solver which uses the same constraint language. The CSPs are in fact related to real life applications. So we are using N Queens problem and a car sequencing problem which are example of CSP to do a comparative study between the different constraint solvers.

The remainder of this paper is presented as follows. We will explain about coverage driven test generation in section 2. In Section 3, we briefly explain the constraint satisfaction problem. We then describe two CSP problems, a car sequencing problem and N Queens problem. Section 4 describes the constraint solvers used in this work. Finally, we present experimental results in Section 5, and give some concluding remarks in Section 6.

2 COVERAGE DRIVEN TEST GENERATION

The main challenge in using random or constraint random verification is that we have to manually analyze the coverage report, find the untested scenarios and modify the test cases to attain 100% coverage. The aim of automated coverage directed test generation is to allow replacement of the above manual effort by an automatic method. There are two benefits that can be achieved by applying CDTG or automated CDTG. The first is that unobserved scenarios will be generated. The second benefit is that certain scenarios can be more easily tested multiple times with different input parameters.

For automated coverage driven test generation first the coverage metric is defined, then the constraints for random test generation. The tests are applied to the design to produce simulation trace and coverage results. From the simulator the coverage report is extracted. The results are analyzed by the tool. Targets that have been missed so far are identified. New test cases and/or constraints will be added to target the coverage holes. This process will be iterated until desired coverage is achieved.

An example where CDTG is applied is in the verification of a processor. Let the design specification be as follows. The processor should be a 32 bit and can handle 10 basic instructions like load, store, add, sub, jump, branch, move, compare equal to, compare less than and compare greater than. Let one of the test case scenario be as follows. The test case should have 25 instructions. It should contain 5 load instructions, 5 store instructions, 2 jump and branch instructions, 3 add and sub instructions. There should be 2 move instructions and 3 compare instructions. There should be 1 immediate instruction in every 5 instruction,1 register source in every 3 instruction,2 register destination in every 5 instruction...etc.

Let us consider another example where we are interested in verifying a particular property. Let the property be the occurrence of data hazard read after write (RAW) in the pipelined processor with five stages. To test data hazards of RAW, we randomly generate instructions which depend on results of prior instructions which are still in the pipeline. The following are the 2 main conditions for this scenario.

- The resource accessed by current instruction is the destination resource of prior instruction.
- The scenarios of data hazards should cover all of pipeline stages.

The first principle ensures that we can generate the condition of data hazards. We can get complete coverage by using the final principle. The above test scenarios are equivalent to constraint satisfaction problem (CSP). So a CSP is used to study the effectiveness of the constraint solver of VCS 2009.06.

3 CONSTRAINT SATISFACTION PROBLEM

Constraint satisfaction problems or CSPs are mathematical problems defined as a set of objects whose state must satisfy a number of constraints or limitations. CSPs are problems we face in our everyday life. The most common examples for CSP in industry are the N Queens problem and car sequencing problem.

3.1 N Queens problem

In the game of chess, a queen can move as far as she pleases, in the same row, in the same column, or diagonally. It is often used as an example problem for testing various programming techniques including constraint solvers. The N Queens problem is a CSP of placing N chess queens on an NN chessboard so that no two queens attack each other. If we assume the chess board as a NxN matrix and dij=1 for the presence of a queen and dij=0 for absence of queen, solving the below mathematical equation gives the solution for the N queens problem.

$$Max \sum_{i=1}^{N} \sum_{j=1}^{N} d_{ij}$$

$$\sum_{i=1}^{N} d_{ij} \leq 1 \quad \forall j = 1, ..., N$$

$$\sum_{j=1}^{N} d_{ij} \leq 1 \quad \forall i = 1, ..., N$$

$$\sum_{i=1}^{N} \sum_{j=1}^{N} d_{ij} \leq 1 \quad where\ i + j = k \ \ \forall k = 2, ...2N$$

$$\sum_{i=1}^{N} \sum_{j=1}^{N} d_{ij} \leq 1 \quad where\ i - j = k \ \ \forall k = 1 - N, ...N - 1$$

$$d_{ij} \epsilon (0, 1) \qquad \forall i, j = 1, ..., N$$

3.2 Car sequencing problem

The car-sequencing problem arises from the manufacture of cars on an assembly line. A number of cars are to be made on a production line; they are not identical because different options are available as variants on the basic model. We cannot have an assembly line to allow sufficient time to put every possible option on every car in the line, since it is very expensive. So the assembly line is modified to predetermined number of cars with a particular combination of requirements to predetermined number of cars without the particular combination of requirements. Let us say the ratio of cars to have air conditioning be 2/3. Then we can say that the assembly line has a capacity constraint of 2 out of 3 for air conditioning.

For our purpose we are taking a small car sequencing problem. The problem is described in the following section. At first we will generate a sequence of 10 cars. Then for the same constraints we will produce car sequences containing 20 cars, 30 cars and 40 cars. This is

done in order to check how the constraint solver will behave if the test case needed to be generated is having large length.

It can be seen that the constraints for the car sequencing problem and that for the test case scenario is similar. For example in test case scenario it was said that there should be 25 instructions which is similar to having 10 cars is the generated sequence for CSP. There were 10 different instructions for the processor which is similar to have 6 types of car in CSP. So it is appropriate to see an actual test case scenario as a CSP problem. The car sequencing problem we consider has the following conditions for a sequence of 10 cars:

Table 1: CAR SEQUENCING PROBLEM FOR A SEQUENCE OF 10 CARS

	Properties					No of Cars
TYPE	A	B	C	D	E	
0	1	0	1	1	0	1
1	0	0	0	1	0	1
2	0	1	0	0	1	2
3	0	1	0	1	0	2
4	1	0	1	0	0	2
5	1	1	0	0	0	2
Max. Capacity	2 out of 3	2 out of 3	1 out of 3	2 out of 5	1 out of 5	

4 CONSTRAINT SOLVER

Figure 2 is a simple representation of how a constraint solver works. The search element is typically depth-first chronological backtracking by default, although a solver will often allow different search algorithms. When searching, a variable and value must be selected. This can be done statically or with a dynamic heuristic. The simplify component contains a queue of constraints which need to be propagated. When a constraint is propagated, and removes values from the variable domains, the domain events cause other constraints to be added to the queue. Propagation of constraints on the queue is iterated until the queue is empty.

4.1 VCS 2009.06

VCS is an industry leading constraint solver. It is powered by multiple solver engines which will simultaneously analyze all user specified constraints. These engines will find a solution to user constraints, if one exists, minimizing constraint conflicts and maximizing verification productivity. VCS is based on SystemVerilog. SystemVerilog has sometimes been called an HDVL, since it combines the strengths of HDLs and HVLs. SystemVerilog is based on the widely used Verilog HDL, but has new functionality for verification and high-level system design. This makes it both powerful and easy to learn. Using the same language for both design and verification also makes it easier to access the internals of the DUT, no special interfaces are needed. We choose three other constraint solvers randomly for our comparison purpose, namely, QUSETASIM, MINION, GECODE and ARTELYS KALIS (commercial tool).

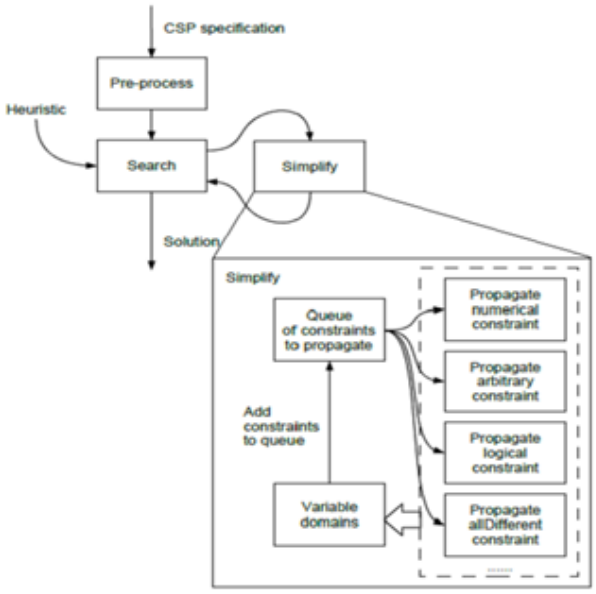

Fig. 2: Block diagram for a constraint solver

4.2 QUSETASIM

SystemVerilog LRM specifies the language only. It does not specify the constraint solver algorithm. So different vendors use different solver engines, which is based on different algorithms. Some vendor tools can solve the constraints quickly while some may not. Mentor's Questa products are based on a single-kernel verification engine that integrates an HDL simulator, a constraint solver, an assertion engine, functional coverage and a common user interface. Coverage-driven test generation (CDTG) is supported by using Questa's high-performance assertion engine, a modern high-performance constraint solver, and extensive functional coverage features.

4.3 MINION

Minion is constraint solver based on interleave splitting (also called branching) and propagation. Propagation is the basic operation of search, and splitting simplifies the CSP instance. The user can view the solution process as the repeated transformation of the CSP until a solution state is reached. The main features are 1) fast and efficient for a wide range of problems 2) fixed implementations of memory management, propagation algorithms 3) tuning only possible by adapting the input file.

4.4 ARTELYS KALIS

Artelys Kalis is a commercially available tool. Artelys Kalis is an open constraint programming environment for solving constraint satisfaction problems through a C++ library. It is based on propagation of constraint and other powerful optimization strategies. Artelys Kalis has been completely designed in an object-oriented programming manner.

4.5 GECODE

Gecode is a toolkit for developing constraint-based systems and applications. Gecode is open for programming. It supports the programming of new propagators (as implementation of constraints), branching strategies, and search engines. New variable domains can be programmed at the same level of efficiency as finite domain and integer set variables that come predefined with Gecode. Gecode is implemented in C++ that carefully follows the C++ standard and can be compiled with modern C++ compilers.

5 EXPERIMENTAL RESULT

In order to generate the solution for the N Queens problem first an array arr[N] of length N is made. The content of the array can be between 1 and N. We simplified the problem constraints for placing the queens on the board into two. The simplified constraints are

1. The content of all the arrays should be different from each other
2. For $i = 1....N, j = i + 1....N$; $arr[i] - arr[j]! = |(i - j)|$

Solving these two constraints gave the solution for the N Queens problem and we obtained the following result.

Table 2: TIME IN SEC FOR SOLVING THE N QUEEN PROBLEM.

	N=5		N=6		N=7	
	Time	Constraint number	Time	Constraint number	Time	Constraint number
VCS	0.008	6	0.017	7	0.023	8
QUESTA SIM	0.01	6	0.139	7	0.143	8
MINION	<0.001	25	<0.001	31	0.0156	45
ARTELY KALIS	< 0.001 (2 soln)	3	< 0.001 (10 soln)	3	< 0.001 (4 soln)	3

The table 2 shows the time required to obtain the results for N Queens problem. We can see that ARTELY KALIS was able to provide all possible results in small time when compared to the other CDTG tools.

For modeling the car sequencing problem; first we assigned each type of car a number for 0 to 5. Then an array of size 10 is defined which should have value from 0 to 5. Similarly we converted all the conditions into corresponding constraints. If a condition is not able to be converted as a single constraint then a group of constraints is used to implement the condition. Once all the conditions are converted to constraints the next step is to randomly generate the sequence of car to fill the array. The constraint solver will generate the random sequence which will satisfy all the applied constraints. Then by changing the array length to 20, 30 and 40 all the other sequences were generated. With our experiment we obtained the following result.

Car sequencing problem is a good example of CSP with a larger number of constraints (constraints>50). From car sequencing problem results we can see that the time requirement

Table 3: TIME IN SEC FOR SOLVING THE PROBLEM.THE x DENOTES THAT THE RESULT WAS NOT OBTAINED

	10 cars	20 cars	30 cars	40 cars
GECODE	0.00005	0.0003	0.003	0.0181
VCS	0.035	0.054	0.085	0.137
QUESTA SIM	0.053	0.096	0.130	0.182
MINION	0.015	8.432	45.058	71.412
ARTELY KALIS	0.010	x	x	x

to generate the sequence increases for VCS and QUSETASIM as the constraint number increases (Table 3). But for the GECODE based solver the time required to generate the solutions where much less when compared to VCS and QUSETASIM.

The table 4 shows the memory consumption for VCS and GECODE based solver for the car sequencing problem. From this we can see that VCS has high memory consumption.

Table 4: MEMORY USED FOR SOLVING THE CAR SEQUENCING PROBLEM

	10 cars	20 cars	30 cars	40 cars
GECODE	49KB	308KB	484KB	1543KB
VCS	5072KB	74000KB	106000KB	126000KB

Uniformity of randomization is important. For example, suppose we have a simple constraint $0 \leq X \leq 10$. If we randomize X a number of times and each time it may return the value 0, this meets the constraint, but we are interested in different values for X. So if we repeat the randomization process several times it doesn't guarantee that a new solution will be generated each time (if there is more than one solution). This will make attaining maximum coverage very difficult.

Hence the problems with the constraint solvers of existing CDTG tools can be summarized as follows:

1. Solving large complex constraint sets is a bottleneck in CDTG due to the large amount of time spent solving these constraint sets.
2. Constraints with large domain of input requires huge amount of memory.
3. Only one solution is generated at a time.
4. There is no guarantee uniformity in randomization.

From car sequencing problem results we can see that the time requirement to generate the sequence linearly increases for VCS and Questasim as the constraint number increases. But for the other tools they have smaller time for small number of constraints, but as the number of constraints increases time required increases exponentially. So VCS and Questasim has better scalability. The same constraint language is used in Questasim and VCS. But Questasim takes more time to produce the result than VCS. This proves that the constraint solver of VCS is a very good solver engine. If contradicting constraints are present constraint solver of VCS specifies which constraints are conflicting. This is very useful for engineers who are using CDTG to get test cases. Based on our experiments we can summarize our findings in table 5.

Table 5: COMPARISON BETWEEN THE TOOLS

	MINION	ARTELYS KALIS	GECODE	VCS 2009.06	QUESTA SIM
Syntax	Small and simple	Simple and rich	Simple and rich	Simple and rich	Simple and rich
Scalability	Poor		Non Linear	Linear	Linear
Code size	Large	Medium	Medium	Small	Small
Conflicting constraints are present	No solution	No solution	No solution	Shows conflicting constraints	No solution

We modeled the same car sequencing problem in SystemVerilog in two methods. In the first method we needed 51 constraints and it took 0.088 seconds to generate the sequence. In the second method we had 23 constraints and needed 0.0350 seconds to generate the sequence. Hence we can see that the number of constraints is related to the time taken to solve the problem. If we were able to implement the condition 'n' out of 'm' sequence should contain 'x' as a single constraint, the number of constraints require to generate the required sequence will be less. This will make the time required to generate the required sequence smaller. But for SystemVerilog there is no instruction to implement the constraint one out of 10 cars should be of type 0.This has to be implemented by using a combination of a number of constraints. For example we specify that if the first car in the sequence is of type 0 then others can't be of type 0, and repeat it for all other combinations.

$$\begin{aligned} constraint\ p44\{(any\_x[0] == 0) \rightarrow \{ & any\_x[1] != 0; \\ & any\_x[2] != 0; \\ & \vdots \\ & \vdots \\ & any\_x[9] != 0\}; \} \end{aligned}$$

But for the other tools they have instructions which will help to specify the above constraint. For example in MINION the following instruction helps to implement the above constraint.

$$\begin{aligned} &discrete\ q[10]\ (0..5) \\ &occurrence\ (q, 0, 1) \end{aligned}$$

6 CONCLUSION

A car sequencing problem and N Queens problem is modeled using VCS 2009.06, Questasim, Minion, Gecode and Artelys Kalis. Then by generating the sequence(s) which satisfies the constraint, the performance of the constraint solvers were analyzed. The results in the previous section show that SystemVerilog is a powerful language for modeling constraints and also the constraint solver of VCS 2009.06 is very powerful. Based on the results we identified the areas which require improvement in order to get better coverage results. In

future we would like to propose a methodology which helps to attain the required coverage with less time and memory consumption.

References

[1] W. Yingpan , *A Coverage-Driven Constraint Random-Based Functional Verification Method of Pipeline Unit,* Computer and Information Science, ACIS International Conference,2009 pp. 1049-1054.

[2] M. Benjamin, *A study in coverage-driven test generation*, Design Automation Conference, 1999. Proceedings. 36th Issue ,1999 pp. 970 - 975.

[3] S. Fine, *Coverage Directed Test Generation for Functional Verification using Bayesian Networks*, Design Automation Conference, 2003. Proceedings Issue Date: 2-6 June 2003pp. 286 - 291.

[4] O. Guzey, *Coverage-directed test generation through automatic constraint extraction*,High Level Design Validation and Test Workshop, 2007. HLVDT 2007. IEEE International Issue Date: 7-9 Nov.2007 pp. 151 - 158

[5] M. Braun, *Comparison of Bayesian Networks and Data Mining for Coverage Directed Verification Category* , Simulation-Based Verification Eighth IEEE International High-Level Design Validation and Test Workshop (HLDVT'03)

[6] H. Shen , *Designing an Effective Constraint Solver in Coverage Directed Test Generation*, Proceedings of the 2009 International Conference on Embedded Software and Systems, 2009, pp. 388-395.