*URL http www elsevier nl locate entcs volume html pages*

*Evaluating Semi Exhaustive Veri cation* Techniques for Bug Hunting

*Ranan Fraer Gila Kamhi Limor Fix Moshe Y Vardi Logic and Validation Technology Intel Corp Haifa Israel*

*Dept of Computer Science Rice University*

*Abstract*

*We propose a methodology to evaluate a rich set of BDD subsetting heuristics with respect to bug hunting and apply it to a set of real life Intel designs Our results illustrate that the evaluation metrics used to rate these heuristics in previous work were not tuned for bug nding e ciency which we believe is the major criterion that the heuristics need to meet*

*Introduction*

*The increasing complexity of today s cutting edge microprocessor designs makes* the detection of functional bugs a much more challenging task Such bugs can be very expensive when caught late only at the post silicon stage therefore there is a strong need to guarantee the absence of hardware design errors before manufacture

*Functional RTL validation is addressed today by two complementary tech nologies The more traditional one simulation has high capacity and full chip* simulation is possible However simulation is rarely exhaustive covering only a tiny fraction out of all the possible behaviors of the design and leaving the door open for subtle bugs to slip in

*This has prompted the use of formal veri cation as an alternative vali dation technique Formal veri cation guarantees full coverage of the entire* state space of the design thus providing high con dence in its correctness Still the more automated and therefore the more popular formal veri cation symbolic model checking has a severe problem of limited capacity State of the art model checkers can hardly verify moderate size designs of the order of hundreds of sequential elements

*In recent years a hybrid approach that merges the strengths of the two* validation technologies has been introduced Semi exhaustive veri ca tion addresses the concerns of practicing veri ers by shifting the focus from veri cation to falsi cation Rather than ensuring the absence of bugs it turns

*c Published by Elsevier Science B V Open access under* [*CC BY-NC-ND license*](http://creativecommons.org/licenses/by-nc-nd/3.0/)*.*

*the veri cation tool into an e ective bug hunter This hybrid approach aims* at improving over both simulation and formal veri cation in terms of state space coverage and capacity respectively

*Semi exhaustive veri cation is not only useful when the design is too large* to be fully veri ed against the speci cation It can also pay o in the early stages of the veri cation where the bugs found are not real design errors but indicate holes in the speci cation In this stage that we call speci ca tion debugging the veri er is in the loop of model checking and speci cation modi cation based on the feedback from the symbolic model checker The turn around time of the model checker becomes then very critical to the pro ductivity of the veri er Therefore the bene t of semi exhaustive veri cation is clearly observed at this stage

*Our usage of semi exhaustive veri cation follows the lines of be ing based on subsetting the frontiers during state space exploration whenever* these frontiers reach a given threshold While previous work focuses on the algorithmic issues in particular on the BDD subsetting heuristics the evalu ation of the various algorithms is based only on the number of visited states Although this metric is interesting as a coverage indicator it does not say anything about the success of this technique as a bug nding tool Covering more states and adventuring deeper in the state space is indeed supposed to increase the chances of bug nding but no experimental data has been provided to back this assumption

*The main contribution of our work is in evaluating a rich set of existing* semi exhaustive algorithms with respect to bug nding We propose an ac curate methodology to evaluate a rich set of BDD subsetting heuristics with respect to bug hunting and apply it to a set of real life Intel designs This con rms our intuition that neither the number of visited states nor the den sity of the approximation are su cient criteria for e ective bug nding It also suggests various ways of combining or improving existing algorithms with respect to bug hunting

*The paper is organized as follows In Section we present an overview of* semi exhaustive veri cation with respect to exhaustive veri cation Section surveys the di erent semi exhaustive techniques that we have evaluated with respect to bug nding The methodology that we deployed to evaluate these techniques is described in Section In Section we interpret the experimental results obtained on a set of real life Intel designs Section concludes by suggesting future directions of research

*Semi exhaustive Veri cation versus Exhaustive Ver i cation*

*A common veri cation problem for hardware designs is to determine whether* every state reachable from a designated set of initial states lies within a spec i ed set of good states referred to as the invariant This problem is vari

*ously known as invariant veri cation or assertion checking*

*Invariant veri cation can be performed by computing all states reachable* from the initial states and checking that they all lie in the invariant This reduces the invariant veri cation problem to the one of traversing the state transition graph of the design where the successors of a state are computed according to the transition relation of the model Moreover traversing the state graph in a breadth rst order makes possible to work on sets of states that are symbolically represented as BDDs This is an instance of the general technique of symbolic model checking

*According to the direction of the traversal invariant checking can be based* on either forward or backward analysis Given an invariant I and an initial set of states Init forward analysis starts with the BDD for Init and uses the Img operator to iterate up to a xed point which is the set of states reachable from Init Similarly in backward analysis the PreImg operator is iteratively applied to compute all states from which it is possible to reach the complement of the invariant

*The primary limitation of both exhaustive forward and backward analysis is* that the BDDs encountered at each iteration commonly referred as frontiers can grow very large leading to a blow up in memory or to a veri cation time out Semi exhaustive veri cation addresses this issue by keeping the size of the frontiers under control More precisely each time the size of the current frontier reaches a given threshold only a subset of the frontier is retained to proceed further

*This heuristic can be regarded as a mixture of breadth rst and depth rst* search that still enjoys the bene ts of the symbolic BDD representation With the risk of missing some reachable states hence the name of semi exhaustive it can allow the veri cation to cover more states and detect bugs that would be hard to reach otherwise

*BDD Subsetting Algorithms*

*As expected the e ectiveness of the semi exhaustive veri cation is very sen sitive to the nature of the algorithm employed for subsetting the frontiers A number of BDD subsetting algorithms have been proposed lately in the* model checking literature Each of them is necessarily a heuristic attempting to optimize di erent criteria of the chosen subset This section surveys the subsetting heuristics that we have used in our experiments together with a brief explanation of the underlying algorithms

*An important class of heuristics takes the density of the BDDs as the crite rion to be optimized where density is de ned as the ratio of states represented* by the BDD to the size of the BDD This relates to the observation that large BDDs are needed for representing sparse sets of states as it is often the case for the frontiers Removing a few isolated states can thus lead to signi cant reductions in the size of the BDDs

*Ravi and Somenzi have introduced the rst algorithms for extracting dense BDD subsets Heavy Branch HB and Short Path SP Heavy Branch*

*HB subsetting counts the size and the number of states represented by each* node of the BDD graph Starting from the root it discards the lighter branch of each node in terms of number of states while subtracting the number of nodes contributed by this branch until the size threshold is reached

*Short Path SP attempts to preserve the short paths of the BDD based on* the observation that such paths contribute many states to the set for relatively few nodes It proceeds by computing the length of the shortest path going through each node evaluating the maximum length of paths that have to be preserved and then discarding nodes with no short paths going through them Independently Shiple proposed in his thesis the algorithm Under Approx

*UA that also optimizes the subset according to the density criterion Like*

*HB and SP this algorithm is also based on discarding some nodes of the BDD However the selection of the nodes to be discarded is based on a more in tricate analysis it computes for each node a lower bound on the increase* in density that would follow from discarding this node Unlike HB and SP that can occasionally decrease the density of the result UA can be run in a

*safe mode An additional parameter quality is used to de ne the minimum*

*improvement in density for the replacement of a node to be acceptable Recently Ravi McMillan Shiple and Somenzi proposed Remap Under*

*Approx RUA as a combination of UA with more traditional BDD minimiza*

*tion algorithms like Constrain and Restrict Rather than simply discarding* a node such algorithms remap it to its brother making their father redundant as well and increasing the sharing of the nodes Like in UA the bene t of replacing is computed for each node but this time for all possible replacement operations the most pro table one being selected at the end

*A combined algorithm is Compress COM which applies rst SP with the* given threshold and then RUA with a threshold of to increase the density of the result Although more expensive the combination of the two algorithms is supposed to produce better results

*The Saturation SAT algorithm is based on a di erent idea Rather* than keeping as many states as possible it attempts to preserve the interesting states In the context of the control states are de ned as the interesting ones The heuristic makes sure to saturate the subset with respect to the control states i e that each possible assignment to the control variables is represented exactly once in the subset In terms of BDDs this is implemented by Lin and Newton s Cproject operator

*Actually the original algorithm of Shiple considered a convex function of the number of the states and nodes but the CUDD implementation that we used optimizes the density*

# *Evaluation of Semi exhaustive Search Techniques*

*This section describes the methodology and the criteria that we deployed to* evaluate the frontier subsetting heuristics with respect to bug nding e ciency

*Evaluation Methodology*

*We selected for the purpose of the evaluation four real life Intel invariant veri cation tasks that indicate bugs in the model or speci cation We gave priority* to the veri cation tasks with long i e at least cycles counter example traces to get rich data with respect to frontier subsetting All veri cation tasks except for ckt have been identi ed to have speci cation bugs In ckt we have planted a design bug for the evaluation purposes

*We report three modes of operation relative to the aggressiveness level* of the approximation namely high medium and low level according to the threshold that triggers the subsetting Highly aggressive approximation cor responds to the setting of the threshold to BDD nodes whereas the medium and low levels corresponds to the setting of the threshold to and BDD nodes respectively

*As for the frontier subsetting we evaluated all the techniques described in* Section in the following conditions

*SP HB and COM have been evaluated as they are with the threshold taking* one of the values mentioned above

*In the case of UA and RUA the additional parameter quality seems to be* at least as important as the threshold We have chosen to evaluate each of them with di erent values for quality

*UA and RUA take the default value quality meaning that only re placements that do not decrease the density are acceptable*

*UA and RUA are even more conservative in taking quality which means that only replacements bringing increase in density are* acceptable

*Conversely UA are RUA are more relaxed taking quality which*

*means that occasionally they can decrease the density but not more than*

*Unlike where SAT is used in conjunction with the control variables our use of SAT saturates the frontier with respect to the speci cation vari ables More precisely each temporal property is translated internally into* a property checker It is the variables of this checker that are used for saturation

*For comparison we have also included in our data the results produced by* fully exhaustive on the y forward model checking denoted as EXH

*Finally note that the heuristics were compared with identical settings of the* model checker with the same initial order le and with no dynamic re ordering

*in order to reduce the external parameters that a ect the results*

*Evaluation Criteria*

*The interpretation of our results will be structured in three stages The rst* stage reports the summary of our results with respect to bug nding This re ects the perspective of a casual user who only has a black box view of the subsetting heuristics

*In a second stage we try to gain more insight by collecting various statistics* on the behavior of the heuristics along the run Such statistics will include

*N the number of approximations*

*AvgT the average time required for an approximation*

*AvgSt the average percentage of states conserved by an approximation*

*AvgNd the average percentage of nodes conserved by an approximation*

*The last two quantities re ect the aggressiveness of each heuristic It is* important to consider them separately rather than looking just at their ratio i e the density One of the main results of our experiments is that density alone is not a su cient condition for success heuristics achieving a very good density can often miss the bug due to their high aggressiveness and vice versa The most accurate analysis is performed in the third stage It assumes

*that backward analysis is possible on the model at hand in order to compute* the set of target states that could lead to a bug This computation starts from the error states that violate the invariant and iteratively applies the PreImg operator till reaching a xed point

*The set of successive frontiers obtained by approximation during the for ward analysis are then graded according to their success in keeping target* states and throwing away other states For this purpose we de ne the follow ing grading function which measures the target states covered by the approx imated frontier as a percentage of a total number of target states

*states approxF rontier T arget*

*Grade approxF rontier*

*states T arget*

*As a by product this evaluation technique illustrates the impact of the* target enlargement technique for bug nding This technique aims at en larging the set of error states violating the invariant with some of the closest target frontiers The intuition is that a larger target increases the chances of hitting a bug faster during on the y forward analysis and consequently reduces the amount of searching that needs to be done Therefore techniques that get a high grade are expected to perform better i e to nd the bug faster and with less memory consumption when applied together with target enlargement

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Ckt* | *ckt latches inputs* | | | | | | *ckt latches inputs* | | | | | | | | |
| *thresh* | *nodes* | | | *nodes* | | | *nodes* | | | *nodes* | | | *nodes* | | |
|  | *R* | *iter* | *time* | *R* | *iter* | *time* | *R* | *iter* | *time* | *R* | *iter* | *time* | *R* | *iter* | *time* |
| *EXH* | *B* |  |  |  |  |  | *B* |  |  |  |  |  |  |  |  |
| *SP* | *B* |  |  | *B* |  |  | *B* |  |  | *B* |  |  | *B* |  |  |
| *HB* | *D* |  |  | *D* |  |  | *B* |  |  | *B* |  |  | *B* |  |  |
| *UA* | *B* |  |  | *B* |  |  | *B* |  |  | *B* |  |  | *B* |  |  |
| *RUA* | *B* |  |  | *B* |  |  | *B* |  |  | *B* |  |  | *B* |  |  |
| *UA* | *B* |  |  | *D* |  |  | *B* |  |  | *B* |  |  | *B* |  |  |
| *RUA* | *B* |  |  | *D* |  |  | *B* |  |  | *B* |  |  | *B* |  |  |
| *UA* | *D* |  |  | *D* |  |  | *B* |  |  | *B* |  |  | *B* |  |  |
| *RUA* | *B* |  |  | *B* |  |  | *B* |  |  | *B* |  |  | *B* |  |  |
| *COM* | *D* |  |  | *D* |  |  | *B* |  |  | *B* |  |  | *B* |  |  |
| *SAT* | *B* |  |  | *B* |  |  | *O* |  |  | *O* |  |  | *O* |  |  |

*Table*

*Summary of results for ckt and ckt*

# *Interpretation of the Experimental Results*

*Stage Summary of Results with respect to Bug Finding*

*Tables compare the bug nding e ectiveness of all the subsetting heuris tics with di erent approximation thresholds In the case of ckt and ckt no* results are reported for the threshold of BDD nodes since the inter mediate frontiers until hitting the bug do not grow that large The letters in the column R stand for the three possible outcomes of the veri cation

*B the bug is found*

*D the veri cation is blocked in a dead end i e a false xed point is* reached

*O a time out or memory out has occurred*

*The other two columns record the iteration in which the bug dead end is* reached or the state explosion occurs and the corresponding CPU time in seconds The numbers for memory consumption are omitted from these tables as they follow the same pattern as the CPU time

*Exhaustive vs semi exhaustive As shown in the EXH line the bug* can also be found by fully exhaustive veri cation in out of the circuits This is due to our selection of veri cation problems where a bug was known to exist Only in ckt in which the bug was arti cially introduced exhaustive veri cation runs out of memory

*Nevertheless most of the approximation heuristics prove themselves by* consistently nding the bug too even in the cases ckt where exhaustive veri cation fails to do so More importantly when successful semi exhaustive algorithms signi cantly outperform the exhaustive one in terms of CPU time

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Ckt* | *ckt latches inputs* | | | | | | *ckt latches inputs* | | | | | | | | |
| *thresh* | *nodes* | | | *nodes* | | | *nodes* | | | *nodes* | | | *nodes* | | |
|  | *R* | *iter* | *time* | *R* | *iter* | *time* | *R* | *iter* | *time* | *R* | *iter* | *time* | *R* | *iter* | *time* |
| *EXH* | *B* |  |  | *B* |  |  | *O* |  |  |  |  |  |  |  |  |
| *SP* | *B* |  |  | *D* |  |  | *B* |  |  | *B* |  |  | *B* |  |  |
| *HB* | *B* |  |  | *B* |  |  | *D* |  |  | *D* |  |  | *D* |  |  |
| *UA* | *B* |  |  | *B* |  |  | *O* |  |  | *O* |  |  | *O* |  |  |
| *RUA* | *B* |  |  | *B* |  |  | *O* |  |  | *O* |  |  | *O* |  |  |
| *UA* | *B* |  |  | *B* |  |  | *B* |  |  | *B* |  |  | *B* |  |  |
| *RUA* | *B* |  |  | *B* |  |  | *D* |  |  | *D* |  |  | *D* |  |  |
| *UA* | *B* |  |  | *B* |  |  | *D* |  |  | *D* |  |  | *D* |  |  |
| *RUA* | *B* |  |  | *D* |  |  | *D* |  |  | *D* |  |  | *D* |  |  |
| *COM* | *D* |  |  | *D* |  |  | *B* |  |  | *B* |  |  | *D* |  |  |
| *SAT* | *B* |  |  | *D* |  |  | *D* |  |  | *D* |  |  | *D* |  |  |

*Table*

*Summary of results for ckt and ckt*

*Impact of the threshold Increasing the aggressiveness of the approxi mation by lowering the threshold to or nodes rarely prevents* the nding of the bug At most the lower threshold causes the veri cation to miss the closest bugs and perform a few more iterations until hitting the next ones Quite surprisingly even then the overall time stays much smaller due to the speed of performing single steps on smaller frontiers

*A quite unusual e ect can be observed in the case of the UA RUA heuris tics for ckt Lowering the threshold from to nodes helps in*

*nding the bug in less iterations This shows that the rst more conservative approximation is not necessarily the one that preserves the interesting buggy* states

*Impact of the quality The quality factor is responsible for the ne* tuning of the UA RUA approximations Lowering the quality from UA to UA and then to UA leads to more aggressive approximations even to the extent of degrading the density of the result in the case of UA However here more aggressive does not necessarily mean better Indeed UA is de nitely worse leading to dead ends in many cases and there is no clear winner between UA and UA

*Comparing heuristics The SP and UA heuristics seem to be the overall* winners both of them nding the bug in out of the cases As we will justify later this is due to their high conservativeness At the opposite side of the spectrum we nd the highly aggressive heuristics COM and SAT that rarely

*nd the bug but when they do they are much faster than SP or UA*

*In the absence of additional data these observations are necessarily spec ulative The next two sections will allow us to make more educated comments* based on the statistics collected along the run and the grading of the approx imated frontiers

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Ckt* | *ckt threshold nodes* | | | | | | | *ckt threshold nodes* | | | | | | |
|  | *R* | *iter* | *time* | *N* | *T* | *St* | *Nd* | *R* | *iter* | *time* | *N* | *T* | *St* | *Nd* |
| *EXH* | *B* |  |  |  |  |  |  | *O* |  |  |  |  |  |  |
| *SP* | *D* |  |  |  |  |  |  | *B* |  |  |  |  |  |  |
| *HB* | *B* |  |  |  |  |  |  | *D* |  |  |  |  |  |  |
| *UA* | *B* |  |  |  |  |  |  | *O* |  |  |  |  |  |  |
| *RUA* | *B* |  |  |  |  |  |  | *O* |  |  |  |  |  |  |
| *UA* | *B* |  |  |  |  |  |  | *B* |  |  |  |  |  |  |
| *RUA* | *B* |  |  |  |  |  |  | *D* |  |  |  |  |  |  |
| *UA* | *B* |  |  |  |  |  |  | *D* |  |  |  |  |  |  |
| *RUA* | *D* |  |  |  |  |  |  | *D* |  |  |  |  |  |  |
| *COM* | *D* |  |  |  |  |  |  | *D* |  |  |  |  |  |  |
| *SAT* | *D* |  |  |  |  | *e* |  | *D* |  |  |  |  | *e* |  |

*Table*

*Statistics collected for circuits ckt and ckt with threshold nodes*

*Stage Statistics Collected Along the Run*

*Due to space limitations we present only the statistics collected in two rep resentative cases the veri cation of ckt and ckt with a threshold of nodes Table reports the corresponding data where the additional column* N stands for the number of approximations while T St and Nd are short names for the averages AvgT in seconds AvgSt and AvgNd in percentages respectively see section

*The data for ckt is characteristic of medium complexity problems where* the bug or the dead end is reached relatively fast and thus only a few ap proximations are required In contrast the veri cation of ckt involves more approximations and its statistics re ect a slightly di erent picture

*Aggressiveness In the previous section we discussed aggressiveness as* a function of the threshold or the quality and its impact on bug nding In a di erent dimension we see here the aggressiveness as a function of the heuristic and we e ectively measure it through the average quantities St and Nd

*UA and RUA seem to be the most conservative i e least aggressive approximations keeping at least of the states due to the high value of* the quality factor Sometimes even too conservative see the data for ckt leading to the same blow up in memory encountered in the fully exhaustive veri cation

*SP is also quite conservative when only a few approximations are involved*

*ckt but on the long run ckt SP gets more relaxed losing as much as*

*of the states In contrast both UA and RUA are more predictable consistently keeping of the states and thus performing better on the* long run

*HB UA and RUA are much less careful often throwing more than*

*of the states and occasionally losing in density this comes as a surprise in* the case of HB whose explicit purpose is to increase the density of the result This is not the case for COM which de nitely achieves the best density throwing away much more nodes than states usually one order more How ever often COM throws away too many states ending up in dead ends We attribute its aggressiveness to the fact that it applies RUA on the result of SP with a threshold of nodes thus allowing an arbitrarily large reduction Even more aggressive is SAT which keeps only a few hundreds of states and nodes This is often bound to fail unless the saturated variables are chosen very carefully or the frontiers are very dense in bugs as it seems to be

*the case for ckt*

*Speed The overall time for nding the bug depends not only on the* quality of the approximations and the number of iterations but also on the time required by the approximations themselves It is this last factor that we measure in the column T

*The numbers here con rm the theoretical predictions HB and SP in this* order are the fastest heuristics their time complexity being linear in the size of the BDD On the other hand both UA and RUA have a quadratic worst case complexity and thus are often signi cantly slower It is this factor that accounts for the di erence in ckt where SP nishes much faster than UA in spite of performing more iterations Also achieving higher quality takes more time as UA is slightly slower than UA which on its turn is slower than UA

*As a combination of SP and RUA COM inherits the quadratic complexity* of RUA Finally SAT is the most expensive approximation based on the data for all the cases and not just Table Indeed its algorithm requires an operation of existential quanti cation at the level of each variable that is not saturated As a consequence there is a high price to pay for the SAT when applied to large BDDs

*Stage Grading the Approximated Frontiers*

*Out of the four examples ckt was the only one for which backward analysis* completes and for which we were able to compute the target states Table reports the data obtained by grading the approximated frontiers for various heuristics with a threshold of nodes The columns R iter time and N are as in Table The next columns contain couples of the form ik Gradek with ik being the iteration of the k th approximation and Gradek the grade of the corresponding approximated frontier

*Comparing SP with HB we note that SP starts better conserving more* target states in the iterations and but HB wins in the iteration the last one before the bug keeping nearly X more target states than SP This accounts for the fact that HB hits the bug in the iteration while SP misses it and eventually enters in a dead end

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Ckt* | *ckt threshold nodes* | | | | | | | |
|  | *R* | *iter* | *time* | *N* | *i Grade* | *i Grade* | *i Grade* | *i Grade* |
| *EXH* | *B* |  |  |  |  |  |  |  |
| *SP* | *D* |  |  |  | *e* | *e* | *e* | *e* |
| *HB* | *B* |  |  |  | *e* | *e* | *e* |  |
| *UA* | *B* |  |  |  | *e* | *e* |  |  |
| *RUA* | *B* |  |  |  | *e* | *e* |  |  |
| *COM* | *D* |  |  |  | *e* |  |  |  |
| *SAT* | *D* |  |  |  | *e* | *e* |  |  |

*Table*

*Grades of approximated frontiers for ckt with threshold nodes*

*UA gets the same grade with HB in the iteration but then it loses more* target states than HB in the iteration and thus misses the closest bug in the iteration Nevertheless it eventually comes back and hits the next bug in the iteration

*Even more interesting is the case of RUA which apparently runs head to* head with HB up to the iteration and still misses the bug in the iteration RUA and HB keep indeed the same number of target states in the iteration

*but not necessarily the same states HB manages to keep some of the*

*states at distance from the bug while RUA throws all of them away This* suggests a re nement of our grading function that would weight the target states according to their distance from the bug i e assigning a higher grade to target states closer to the bug

*Finally SAT and COM get the lowest grade at the iteration both losing* too many target states from the very rst approximation Moreover the few target states that remain are not close enough to the bug so advancing a few more iterations does not su ce for reaching any of the buggy states

*Conclusions and Future Work*

*Previous studies advocate the merits of semi exhaustive techniques* on the basis of their coverage of the reachable state space and the density of the approximated frontiers The main contribution of our work is to evaluate these techniques with respect to bug nding Our rst results con rm the potential of semi exhaustive veri cation as a bug hunting tool

*Beyond the point of view of a casual user we also try to gain more in sight into the behavior of the semi exhaustive algorithms For this purpose we collect statistic data on the veri cation runs and analyze it along several* dimensions This analysis provides a better understanding of the various sub setting heuristics and in particular of the tradeo s to be considered for each heuristic

*A shortcoming of most the subsetting heuristics is that they are hard to* tune for bug nding since they are not guided in any way by the speci cation

*that is checked The only exception in this sense is the SAT heuristic but* our results with SAT were quite disappointing It might be that our choice of using SAT with the speci cation variables is not the best one but we rather tend to blame the algorithm of SAT which makes too strong assumptions on the design that is veri ed We believe that re ning the SAT algorithm in this direction as also suggested in would be bene cial with respect to bug hunting

*Acknowledgements We would like to thank Amitai Irron for his great* work on architecting and developing the formal veri cation platform in which we made the evaluation Boris Ginsburg s work on the evaluation of dense frontier subsetting with respect to covered states has inspired this work

*References*

*K Ravi F Somenzi High Density Reachability Analysis In Proceedings of International Conference on Computer Aided Design Santa Clara CA November*

*M Ganai A Aziz E cient Coverage Directed State Space Search In Proceedings of IWLS*

*J Yuan J Shen J Abraham and A Aziz On Combining Formal and Informal Veri cation In Proceedings of the Computer Aided Veri cation Conf July*

*C Yang D Dill Validation with Guided Search of the State Space In Proceedings of DAC*

*A Aziz J Kukula T Shiple Hybrid Veri cation Using Saturated Simulation In Proceedings of DAC*

*R Bryant Graph based Algorithms for Boolean Function Manipulations IEEE Transactions on Computers C August*

*K L McMillan Symbolic Model Checking Kluwer*

*T R Shiple Formal Analysis of Synchronous Circuits PhD thesis University of California at Berkeley*

*K Ravi K L McMillan T R Shiple F Somenzi Approximation and Decomposition of Binary Decision Diagrams In Proceedings of DAC*

*O Coudert J Madre A Uni ed Framework for the Formal Veri cation of Sequential Circuits In Proceedings of ICCAD*

*B Lin R Newton Implicit Manipulation of Equivalence Classes Using Binary Decision Diagrams In Proceedings of ICCD*