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

*Approximate Symbolic Model Checking using* Overlapping Projections

*Shankar G Govindaraju Computer Systems Laboratory*

*Stanford University Stanford CA USA*

*David L Dill Computer Systems Laboratory*

*Stanford University Stanford CA USA*

*Abstract*

*Symbolic Model Checking extends the scope of veri cation algorithms that can be handled automatically by using symbolic representations rather than explicitly searching the entire state space of the model However even the most sophisti cated symbolic methods cannot be directly applied to many of today s large designs because of the state explosion problem Approximate symbolic model checking is an attempt to trade o accuracy with the capacity to deal with bigger designs This paper explores the idea of using overlapping projections as the underlying ap proximation scheme The idea is evaluated by applying it to several modules from the I O unit in the Stanford FLASH Multiprocessor and some larger circuits in ISCAS benchmark suite*

*Introduction*

*The ability to enumerate the set of states reachable from a certain state and the ability to enumerate the set of states that can reach a certain state* are essential to many model checking algorithms Binary Decision Diagrams

*BDDs have proved to be a viable data structure for doing symbolic reach*

*ability on larger hardware designs than before However for many large design* examples even the most sophisticated BDD based veri cation methods can not produce exact results because of size blowup However required properties

*This work was supported by DARPA contracts DABT C and DABT C*

*The content of this paper does not necessarily re ect the position or the policy of the Government and no o cial endorsement should be inferred*

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

*of a design rarely rely on every implementation detail of the design so ap proximate veri cation algorithms may yield meaningful results while handling* larger designs

*We are interested in safety properties that hold for every member of a set* S of states A superset Sap of S is called an overapproximation of S Although Sap may be larger than S it may also have a smaller representation so the computation of Sap may be more e cient than S If every state in Sap satis es a property we can be sure that every state in S also satis es the property Hence a su ciently accurate approximation can yield a useful result

*The approximation used is based on overlapping projections of sets of* states A set of states is represented by a list of BDDs each element of the list constrains possibly overlapping subsets of the state variables The projection of a set S of bit vectors onto a set of one bit variables wj is the

*larger set of bit vectors that match some member of S for all variables in*

*wj the values of other variables are ignored S can be approximated by* projecting it onto many di erent subsets of the variables and considering Sap to be the intersection of all of the approximations

*The idea is evaluated on several control modules from a real large design* unit in the Stanford FLASH Multiprocessor with promising results Proper ties in the design were either shown to hold for all reachable states or actual violations were proved to exist in the exact reachable state space some vio lated assertions resulted from omitting constraints on the possible inputs to the design

*Related Work*

*At a high level this work is quite similar to that of Wong Toi et al who used successive forward and backwards overapproximations and under approximations to verify real time systems That work used polyhedra for* representing sets of real numbers along with BDDs but approximation was used only for the polyhedra not for the BDDs

*Various approaches to approximate reachability and veri cation using BDDs* have preceded this work Ravi et al use high density BDDs to compute an underapproximation of the forward reachable set Cho et al proposed symbolic forward reachability algorithms that induce an overapproximation They partition the set of state bits into mutually disjoint subsets and do a symbolic forward propagation on individual subsets Cabodi et al com bine approximate forward reachability with exact backward reachability Lee et al propose tearing schemes to do approximate symbolic backward reachability They also partition the set of state bits into mutually disjoint subsets They form the block sub relations for the various subsets and then incrementally stitch the block sub relations together until the approximated next state relation is accurate enough to prove or disprove a given property In contrast to the approaches in we are interested in computing overap

*proximations supersets In contrast to the approaches in we allow* for overlapping subsets as overlapping projections have been shown to be a more re ned approximation compared to earlier schemes based on disjoint partitions

# *Background*

*We analyze synchronous hardware given as a Mealy machine M hx y q ni where x fx xkg is the set of state variables and y is the set of input* signals We will use x fx x g to denote the next state versions of

*k*

*the corresponding variables in x fx xkg The set of states is given by*

*x B where B f g The initial state q x B The next state* function is n x B y B x B

*In our applications sets can be viewed as predicates since we can form the* characteristic function corresponding to a set BDDs can be used to represent predicates and manipulate them For example let R x be a predicate with support in x we can compute the image of R under n as

*Im R x n x y x x y x n x y R x*

*Let g be a user speci ed property and g denote the complement of g Then* the preimage of g x ie the set of states that can reach a state violating the property g in one step can be computed as follows

*P re g n x x y x n x y g x*

*Approximation by Projections*

*Let w w wp be a collection of not necessarily disjoint subsets of x*

*Each subset will be referred to as a block We de ne the operator j R which projects a predicate R x onto the variables in wj Let z consist of all* of the Boolean variables in x that are not in wj We can de ne j as

*j R z wj wj z R z wj*

*Clearly the set of Boolean vectors satisfying R is a subset of those satisfying*

*j R This can be written using logical implication as R j R The* projection operator projects a predicate R x onto the various wj s and its associated concretization operator conjoins the collection of projections

*R x R p R*

*p*

*R Rp Rj*

*j*

*Lemma For every predicate R x and collection of subsets w wp of x R R*

*The proof for this lemma is simple since R j R for all j Thus projecting a* predicate R onto a collection of subsets and then concretizing the projections

*by results in an overapproximation*

*It is interesting to note that the pair of functions form a Galois* connection between the partially ordered set describing the concrete space

*x B and the poset describing the abstract space P w B P wp B v where P S denotes the power set of S and the ordering* relation for the abstract space is de ned as R Rp v S Sp i

*i p Ri Si*

*Let R R Rp and S S Sp be two tuples of equal size We de ne the meet u and join t operator between R and S as follows*

*R Rp u S Sp R S Rp Sp*

*R Rp t S Sp R S Rp Sp*

*Given the ordering relation v in the abstract domain it is easy to verify that the join operator returns the least upper bound and meet returns the greatest lower bound of the two elements R and S in the abstract domain Further*

*R S R t S which makes the join operator an approximation of* set union However the meet operator is an exact set intersection operator since R S R u S

*The operator allows us to represent a big BDD with support in x by* a tuple of potentially smaller BDDs with limited support at the cost of loss of accuracy can potentially result in a bigger BDD with bigger support hence we would like to avoid computing R Rp explicitly Let Imap

*the subscript ap denotes approximate return the projected version of the*

*image of an implicit conjunction of BDDs and let P reap return the projected* version of the preimage of an implicit conjunction of BDDs

*Imap R n Im R n x y P reap R n P re R n x y*

*Using Imap we can compute an overapproximation F wdReachap q of* the reachable states for a machine M Analogously using P reap we can com pute an overapproximation BackReachap g of the set of states in M that can reach the set of states g as follows

*F wdReachap q lfp R q t Imap R n BackReachap g lfp R g t P reap R n*

*where lfp is a least xed point iteration which starts with R and on each iteration joins the current approximate set with the approximate* successor set Finally after reaching convergence it returns a tuple R to F wdReachap q or BackReachap g as the case may be The approximate set of states that can be reached is the implicit conjunction F wdReachap q The approximate set of states that can reach g is is the implicit conjunction

*BackReachap g*

*Using Lemma and monotonicity of Im and P re functions it can be* shown that the derived functions Imap and P reap have the property

*Im R x n Im R x n Imap R x n*

*P re R x n P re R x n P reap R x n*

*The proof that F wdReachap and BackReachap are overapproximations su persets follows trivially These operators give us exact results in the special* case when there is just one subset w x in the collection w

# *Overlapping Projections*

*Our scheme for choosing the collection of subsets is presently manual Of* course it would be desirable to automate fully or partially the choice of subsets and we are working on developing good heuristics to do so Our present heuristic tries to put interacting nite state machines FSMs together in one subset Often a master FSM communicates with a number of other slave FSMs This is captured by having blocks where the master is paired with each of its slaves in di erent blocks Occasionally two rather big state machines have a small interface which can be captured by adding bits through which the two machines communicate to the subsets having the corresponding FSMs

*Computing Imap by Multiple Constrain*

*The key step in our approximate forward propagation is computing Imap Imap R n S Sp Im R n x y*

*We would like to be able to compute the Sj s separately without comput ing Im R n Clearly Sj can only depend on the next state functions* of the variables appearing in the jth block wj in w Let j n be the subset of predicates determining the next state for the bits in wj Clearly Sj Im R j n

*To avoid unnecessary BDD blowup we want to avoid the explicit conjunc tion R Sj can be computed by forming the next state relation for block* wj and using early quanti cation However this did not work when we tried it on our larger examples Instead Coudert and Madre have shown how to compute the image of a Boolean function vector using the generalized cofactor also called constrain operator f g x has the same value as f x when g x holds and usually results in a smaller BDD than f

*Coudert and Madre show that Im R j n Im j n R To avoid computing the large BDD for R it is tempting to compute j n R R Rp This works well if the supports of Ri s are disjoint However since we have overlapping subsets the naive method is incorrect Instead for overlapping projections we use the method of multiple con*

*strain Let z zp be dummy state bits with corresponding next state*

*functions R Rp The multiple constrain method relies on the following* key observation

*Im R Rp j n Im j n R Rp z z zp*

*We can optimize on the usual recursive co domain partitioning algorithm by avoiding computing the parts of the range that will be discarded The al gorithm Immc described below implements the required function Imap A* more detailed treatment is given in

*function Immc R Rp n nm v n nm R Rp*

*for j p down to by do v v v m j*

*endfor*

*return Im fv v m g*

*Computing P reap by Domain Cofactoring*

*The key step in our approximate backward propagation is computing P reap P reap R n S Sp P re R n x y*

*Instead of using next state relations to compute the preimage Filkorn showed that the the preimage of a set represented by a BDD Q can be ob tained by substituting the state variables in Q with their corresponding next* state function The obvious algorithm to compute Sj would be to substitute the functions in R and then hide existentially all the variables apart from those appearing in wj However since most of the variables would be hidden the size of the intermediate BDD during this computation would be prohibitive even when the nal BDD was small

*Instead Sj is computed by recursively cofactoring on the domain variables* in wj which allows the existential quanti cation to be done on the y Each state variable x in R is renamed to x to avoid con icts Let be a map from each x to the function that is to be substituted for it Initially maps

*i*

*xi to its next state function but is modi ed in the recursive calls to the*

*preimage function Only some of the functions in will be used because some*

*i variables do not appear in any Ri let j j be the number of functions in*

*x*

*that will actually be substituted*

*The recursive algorithm P redc the subscript dc denotes domain cofac toring takes as arguments the current substitution the current approxi mation R the approximate reachability set from the rst forward pass I and* the set of variables wj to project onto I is used to prune preimage states that are de nitely not reachable P redc implements the required function P reap

*A more detailed treatment is given in*

*function P redc R Rp I Ip wj if I or or Ip return if j j return R R Rp*

*v next variable from wj to cofactor on*

*t P redc v R v Rp v I v Ip v wj e P redc v R v R v I v I v wj*

*result ite v t e return result*

*The following optimizations reduce the number of recursive calls to P redc*

*If at any point the support of a function in is wholly contained inside* wj it is immediately substituted into the Ri s and thereafter removed from

*When j j all the the support of all Ri s is contained in wj so the* algorithm returns their explicit conjunction

*After cofactoring on variables in wj the support of the functions in is* disjoint from wj hence the result of P redc is either or Since by this point in the recursion the BDDs are generally small the algorithm does the substitution and returns only if the resulting BDD is not a constant This approach worked ne on all the examples that were tested however in case of BDD blowup the algorithm could conservatively return

# *Using Auxiliary Variables to re ne Im and P reap*

*ap*

*The previous schemes can be further improved upon by augmenting the set* of state variables with some auxiliary state variables An auxiliary variable is an internal state component that is added to the implementation with out a ecting the externally visible behavior The idea of augmenting a legal implementation with some extra state components in a way that places no constraints on the behavior of the implementation is not entirely new Abadi and Lamport introduced a special class of auxiliary variables history and prophecy variables to broaden the applicability of re nement mapping tech niques We use auxiliary state variables to broaden applicability of ap proximate reachability techniques

*Converting Internal Wires to Auxiliary State Variable*

*We look for important internal conditions in the combinational logic and con vert them to auxiliary variables An auxiliary variable is useful because it* captures important properties of many state variables into a single new state bit This can be added to the other subsets to capture correlation between many state variables even as the number of variables in di erent subsets is small

*We make use of auxiliary variables by converting them to state variables To assign a next state function to an auxiliary variable we get the fanin cone* for the internal wire it corresponds to A fanin cone of a wire is obtained by topologically moving back from the wire and grabbing all the logic that feeds to it until we hit a op boundary or an input boundary Let f x be the Boolean function for cone of logic feeding into a wire called foo Recall that n is the next state functions for the usual state variables x The next state function for auxiliary state variable foo is obtained by substituting the corresponding next state function from n for each state variable in the support of f x This

*has the e ect of retiming the internal wire foo The initial condition for* auxiliary state variable foo is set by the image computation Im q f This construction is possible for only those internal wires whose fanin cones involve just state variables and no inputs

*This limitation can be circumvented by including the inputs as part of the* state as in a Kripke structure We never used this for any of our results here but the Mealy machine M hx y q ni can be transformed to M

*hx y q n i where x x y and q q The y component is a set with a*

*primed version for each variable in y The next state function for the x state* variables remains the same but for the y variables it is the corresponding input variable from y Assuming totally unconstrained input environment M and M allow the same externally visible behaviors However M allows us more exibility in choosing auxiliary state variables

*Our scheme for choosing which internal abstractions to convert to auxiliary* state variables is presently manual and relies on being able to inspect the RTL source We believe it helps to look at the RTL source because designers often create internal abstractions themselves while coding up their design using a hardware description language such as Verilog Hence we can take leverage o this high level information directly by inspecting the RTL description We presently look for internal wires in the RTL description that have many state variables in their fanin support More details on our heuristic can be obtained from

*Re nement*

*An overapproximation of the states that lie on a path from the initial state q to a state not satisfying a user speci ed property g is computed by repeated* forward and backwards passes until the approximation no longer improves

*function BackAndForth g Rf*

*Rb while Rf Rb do*

*Rf lfp R q t Imap R n u Rb*

*if Rf g return no errors*

*Rb lfp R g t P reap R n u Rf if Rb q return no errors*

*endwhile return Rf*

*The tests Rf g and Rb q can be performed without com*

*puting the explicit conjunctions of the BDDs in Rf and Rb by computing* images using the method of multiple constrain Rf g holds i Im R g f g and R q i Im R q f g If BackAnd Forth is unable to prove the desired property g it is often possible to run it

*again with larger blocks of variables in w*

*Counterexamples*

*If BackAndForth reports a possible error it is useful to check whether there* is an actual error by generating an example path from q to a state that does not satisfy g This both con rms the existence of an error and provides debugging information to the user In exact reachability analysis if an error state is reachable from an initial state it is straightforward to construct a speci c path from the initial state to an error But in approximate analysis such a path may not exist More subtly the algorithm may have found a real error via a non existent path A simple search method was implemented for counterexample generation which worked well on examples

*Starting from the error states the algorithm computes approximate preim ages and stores the preimages obtained at the various iterations of the xpoint* algorithm in a stack Let T T Tm where Tm intersects with the error states be the nal contents of the stack and let Ti be the rst level at which the approximate preimage intersects with the initial state q Choose a single state s from the intersection q Ti and compute an exact image of s If the image of s intersects with Ti choose a single state s from the intersection and continue moving forward It is also possible that the image of some state sl in layer Tj may lie entirely in Tj and not intersect with Tj at all implying Tj is approximately reachable from sl but not exactly reachable from sl in which case randomly choose another state sl from the image of sl and continue trying to move to the next layer in the stack If the algorithm spends more than steps at the same layer it aborts and reports that it could not

*nd a counterexample*

*This simple algorithm has worked well on proving local safety properties* over the individual submodules of FLASH I O but often fails when we prove global safety properties over the complete design We are currently working on improving this and looking for ways to improve the approximations when the counterexample generation gets stuck

*Experiments*

*The experimental implementation of the method was in LISP calling David* Long s BDD package implemented in C via the foreign function interface The method was evaluated on a collection of control circuits from the MAGIC chip a custom node controller in the Stanford FLASH multiprocessor For comparison with earlier work we also present our results when applied to the ISCAS benchmark suite

*Approximate Forward Reachability In the case of s circuit from the* ISCAS benchmark suite earlier approximate schemes based on disjoint partitions resulted in a superset with a satisfying fraction of e

*whereas our scheme with overlapping projections resulted in a tighter superset* with a satisfying fraction of e which represents an improvement by

*e Similarly in case of s results with overlapping projections* were better by a factor of e A more detailed listing of the results we obtained on the other circuits from the ISCAS suite and the results on the FLASH I O modules is given in Further on adding auxiliary state variables the results obtained by overlapping projections over the usual state variables alone was further improved by at least an order of magnitude More details on the results obtained with auxiliary state variables are in

*Approximate Forward and Backward Reachability We applied our approxi mate forward and backward routines to prove some designer provided invariant* properties on various submodules in FLASH I O Out of properties the approximation scheme was able to prove of them and present counterex amples for the remaining More details on the results with the modules in FLASH I O can be obtained from

*Proving global properties on a big design We have also applied our al gorithm to prove some more global properties over FLASH I O Using the* lossless cone of in uence reduction we are able to reduce the original design

*nearly state variables to the order of state variables By doing ap proximate reachability over these variables using overlapping projections we have been able to prove global invariants and disprove others with a* valid counterexample However there is still more to be done before designs of this size can be directly handled by our model checker

*References*

*Abadi M and Lamport L The Existence of Re nement Mappings LICS pp July*

*Bryant R E Graph Based Algorithms for Boolean Function Manipulation IEEE Transactions on Computers Vol C No pp August*

*Burch J R Clarke E M McMillan K L Dill D L and Hwang L J*

*Symbolic Model Checking States and Beyond LICS pp*

*Cabodi G Camurati P and Quer S Symbolic Exploration of Large Circuits with Enhanced Forward Backward Traversals EURO DAC pp*

*Cho H et al Algorithms for Approximate FSM Traversal Based on State Space Decomposition IEEE TCAD Vol No pp December*

*Coudert O and Madre J C A Uni ed Framework for the Formal Veri cation of Sequential Circuits ICCAD pp*

*Cousot P and Cousot R Abstract Interpretation A uni ed lattice model for static analysis of programs by construction or approximation of xpoints POPL pp ACM Press*

*Dill D L and Wong Toi H Veri cation of Real Time Systems by Successive Over and Under Approximation CAV pp*

*Filkorn T Functional Extension of Symbolic Model Checking CAV pp*

*Govindaraju G S Dill D L Hu A J and Horowitz M A Approximate Reachability with BDDs Using Overlapping Projections DAC pp*

*Govindaraju G S and Dill D L Veri cation by Approximate Forward and Backward Reachability ICCAD pp*

*Govindaraju G S Dill D L and Bergmann J P Improved Approximate Reachability using Auxiliary State Variables DAC to appear*

*Kuskin J et al The Stanford FLASH Multiprocessor ISCA pp*

*Lee W Pardo A Jang J Hachtel G and Somenzi F Tearing Based Automatic Abstraction for CTL Model Checking ICCAD pp*

*McMillan K L A Conjunctively Decomposed Boolean Representation for Symbolic Model Checking CAV pp*

*Ravi K and Somenzi F High density Reachability Analysis ICCAD pp*