Fine-Grain Conjunction Scheduling for Symbolic Reachability Analysis

HoonSang Jin[1]*, Andreas Kuehlmann[2], and Fabio Somenzi[1]**

[1] University of Colorado at Boulder
{Jinh,Fabio}@Colorado.EDU
[2] Cadence Berkeley Labs
kuehl@cadence.com

Abstract. In symbolic model checking, image computation is the process of computing the successors of a set of states. Containing the cost of image computation depends critically on controlling the number of variables that appear in the functions being manipulated; this in turn depends on the order in which the basic operations of image computation—conjunctions and quantifications—are performed. In this paper we propose an approach to this ordering problem—the conjunction scheduling problem—that is especially suited to the case in which the transition relation is specified as the composition of many small relations. (This is the norm in hardware verification.) Our *fine-grain* approach leads to the formulation of conjunction scheduling in terms of minimum max-cut linear arrangement, an NP-complete problem for which efficient heuristics have been developed. The cut whose width is minimized is related to the number of variables active during image computation. We also propose a clustering technique that is geared toward the minimization of the max-cut, and pruning techniques for the transition relation that benefit especially from the fine-grain approach.

1 Introduction

Reachability analysis computes the set of states of a state-transition system that are reachable from a set of initial states. Besides explicit methods [18] for traversing states one by one and SAT-based techniques [1] for deciding distance-bounded reachability between pairs of state sets, symbolic methods [9,5] are the most commonly used approach to this problem. Symbolic methods employ BDDs for two purposes: (1) to collect the set of reachable states for deciding when a fixpoint is reached, and (2) to represent the systems' transition relation. Without loss of generality, we will limit our description to forward state exploration. Because of their symmetry, all methods presented in this paper are equally applicable to backward state traversal.

Each traversal step consists of an image computation that calculates the set of states reachable in one transition from a set of current states. For this purpose, the BDD representing the transition relation is conjoined with the BDD of the current states. This is followed by an existential quantification of the current state variables to eliminate the

* Work performed in part while the author was at Cadence Berkeley Labs.

** Work supported in part by SRC contract 2001-TJ-920 and NSF grant CCR-99-71195.

J.-P. Katoen and P. Stevens (Eds.): TACAS 2002, LNCS 2280, pp. 312–326, 2002.

origin information, and a renaming step to re-encode the state set in terms of the current-state variables. Except for small systems, the transition relation can seldom be stored as a single BDD. Instead, it is represented in a partitioned manner. The conjunctively partitioned transition relation [4] is the most common form and consists of a set of *conjuncts*. The advantage of the partitioned form is that the image step can be performed step-wise by a series of AND operations between the current-state BDD and the individual conjuncts of the transition relation. This allows the application of early quantification to eliminate variables as soon as they become dead, i.e., they become unreferenced by any of the following conjuncts. This significantly alleviates the computational complexity of the image computation step.

The partitioned implementation of the image computation step poses an optimization problem with two interdependent goals: (1) to find a partitioning of the transition relation that has small BDD representations for the individual conjuncts, and (2) to determine a conjunction and early quantification schedule that minimizes the maximum BDDs size of the intermediate results. This problem is referred to as the *conjunction scheduling problem*; a simplified version of it is NP-complete [14].

Traditional approaches to the conjunction scheduling problem are based on coarse-grain methods [12,17,15,8]. Their motivation comes from the incentive to avoid the large number of intermediate variables that are needed for a finer grain partitioning. Coarse-grain methods start from large conjuncts, typically entire next-state functions, and try to further cluster them with the goal of finding a schedule that can eliminate a maximum number of variables as early as possible. Only if the BDD for a next-state function cannot be built within a given memory limit, cutpoints are applied to partition its clusters [2]. In these methods the insertion of cutpoints is driven by BDD size limits only and does not take into account its effect on the clustering result.

In this paper we propose a different approach to the clustering and scheduling problem. Instead of starting from large clusters that represent significant fractions of the next-state functions, we begin the process with a fine-grain partitioning based on single gates or small fanout-free sub-circuits. The resulting large number of conjuncts is then carefully clustered with the objective to *minimize the maximum number of variables* that are alive during image computation, and to further *eliminate as many variables as possible* by making them local to single conjuncts. This variable elimination process can also be used to remove state variables, which in turn prunes the corresponding registers from the transition relation.

2 Motivation

We motivate the fine-grain approach of this paper by demonstrating the effect of different clusterings on the complexity of the transition relation and quantification schedule. We use a logic circuit example as it best demonstrates our intention; however, the proposed approach is equally applicable to structures that model other systems. Fig. 1(a) shows a sequential circuit with two primary inputs, one primary output, and three registers storing the circuit state. Part (b) gives the corresponding state-transition graph of the circuit with an initial state described by the predicate $I = \neg x_1 \wedge \neg x_2 \wedge \neg x_3$.

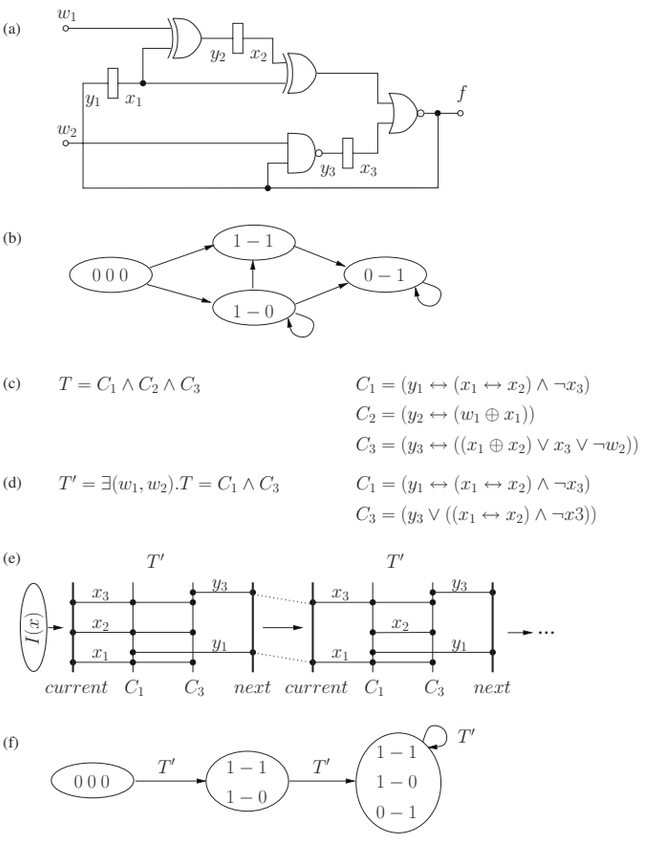


Fig. 1. Example to illustrate the impact of clustering on register pruning: (a) circuit structure, (b) state-transition diagram, (c) original partitioned transition relation, (d) transition relation after one pruning step, (e) quantification schedule used for reachability analysis, (f) state sets visited during breath-first forward traversal

The transition relation T is constructed using one pair of variables for each state holding element. In the example, $\{x_1, x_2, x_3\}$ and $\{y_1, y_2, y_3\}$ denote the *current-state variables* and *next-state variables*, respectively. For the input variables and individual conjuncts of the partitioned transition relation we use $\{w_1, w_2\}$ and $\{C_1, C_2, C_3\}$, respectively. The transition relation T for the given circuit is shown in Fig. 1(c).

Since the primary input variables w_1 and w_2 occur only locally within C_2 and C_3, respectively, T can immediately be pruned by existential quantification resulting in T' (Fig. 1(d)). The elimination of w_1 causes C_2 to become a tautology, which can be removed from T. As a result the current-state variable x_2 is no longer driven by its next-state variable y_2 and becomes unconstrained. This effectively removes the corresponding register from reachability analysis because x_2/y_2 are not involved in the renaming step

and are also not part of the BDD recording the set of reachable states. It should be noted that this differs from removing peripheral λ-latches [6] because the register x_2/y_2 feeds other gates. Since the removal is caused by the disappearance of a next-state variable, we will refer to this techniques as *backward register pruning*. In this example, the variable x_2 cannot be further pruned because it occurs in two conjuncts. However, in general pruning can be applied in multiple iterations.

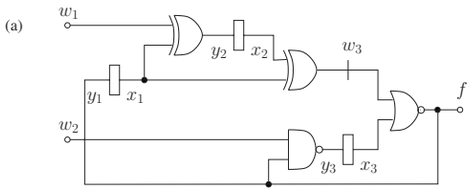
Fig. 1(e) gives the resulting quantification schedule using the pruned transition relation T'. As shown, in the first image computation step x_2 must be constrained by the set of initial states. In the following steps x_2 remains local to T', establishing a relation between C_1 and C_2 only. Note that in this example, only one backward pruning step was applied, thus the transition relation for the first and following image step is identical. However, in general, multiple backward pruning iterations, interleaved with *forward pruning* (as described later), can successively remove more registers. Generally, a transition relation that has been pruned by i forward pruning iterations must not be used before image computation step $i+1$, otherwise unreachable states could get introduced in the reachability results. For the given example, Fig. 1(f) depicts the individual sets of reached states as they are encountered during breadth-first forward traversal.

A different clustering of the transition relation can further reduce the size of its conjunctive form. Similar to the complete elimination of w_1 and w_2, additional variables can be removed by existential quantification *if they are made local to a single conjunct*. Fig. 2(a) shows the modified circuit example with the additional variable w_3 for partitioning the conjuncts C_1 and C_3 and part (b) gives the updated transition relation. Besides C_1, C_2, and C_3, which express the next-state relations of the registers, it also includes the conjunct C_4, which relates the variable w_3 to its driving function.

This finer grain partitioning provides more freedom to iteratively quantify variables. As shown in Fig. 2(c-e), multiple pruning steps can successively reduce the transition relation until it contains only a single, simple conjunct. First, the existential quantification of w_1 and w_2 eliminates C_2 resulting in T'. In contrast to the clustering applied in the previous case, here x_2 becomes now local to C_4 and can be quantified in the next pruning step resulting in T''. As explained above, T'' cannot be applied for the first image computation since at that point x_2 must be constrained by the initial states. Fig. 2(f) shown the quantification schedule to be used for exact reachability analysis. After the initial application of T' all following steps can use the pruned relation T''.

Similar to backward pruning, *forward register pruning* denotes the removal of registers based on the disappearance of current-state variables. In the given example, forward pruning can be applied to remove register x_1/y_1 since x_1 has disappeared from T'' as a result of the previous pruning step. In other words, the absence of the current-state variable x_1 in T'' makes the actual binding of the next-state variable from the previous image step superfluous. Thus y_1 can be removed from T'' by existential quantification, which in turn eliminates C_1. As a result w_3 becomes local to the only remaining conjunct C_3 and can also be eliminated producing T'''.

Due to the existential quantification of y_1, the use of T''' for forward image computation results in an over-approximation of the set of resulting states. The corresponding effect is demonstrated in the top part of Fig. 2(h). As shown, the application of T''' for the second traversal step produces, among others, the unreachable state $(0, 1, 0)$. Note



(b) $T = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

$C_1 = (y_1 \leftrightarrow (\neg w_3 \wedge \neg x_3))$
$C_2 = (y_2 \leftrightarrow (w_1 \oplus x_1))$
$C_3 = (y_3 \leftrightarrow (w_2 \vee x_3 \vee \neg w_2))$
$C_4 = (w_3 \leftrightarrow (x_1 \oplus x_2))$

(c) $T' = \exists(w_1, w_2).T = C_1 \wedge C_3 \wedge C_4$

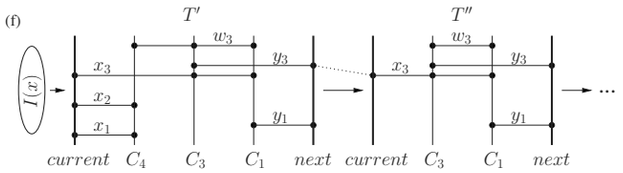
$C_1 = (y_1 \leftrightarrow (\neg w_3 \wedge \neg x_3))$
$C_3 = (y_3 \vee (\neg x_3 \wedge \neg w_3))$
$C_4 = (w_3 \leftrightarrow (x_1 \oplus x_2))$

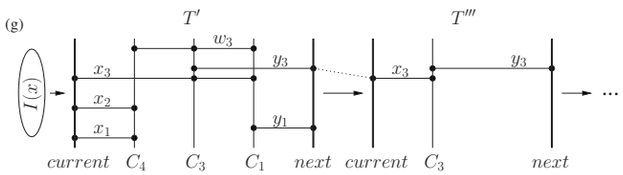
(d) $T'' = \exists x_2.T' = C_1 \wedge C_3$

$C_1 = (y_1 \leftrightarrow (\neg w_3 \wedge \neg x_3))$
$C_3 = (y_3 \vee (\neg x_3 \wedge \neg w_3))$

(e) $T''' = \exists w_3.(\exists y_1.T'') = C_3$

$C_3 = y_3 \vee \neg x_3$





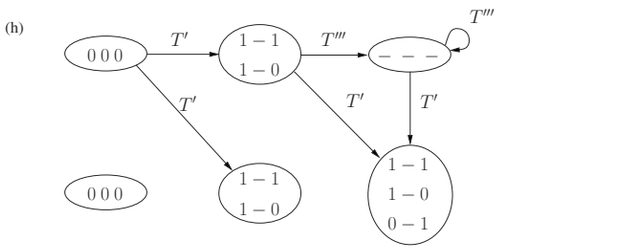


Fig. 2. Example of Fig. 1 using a different partitioning: (a) circuit with additional cut variable w_3, (b) original transition relation, (c-e) transition relation after multiple pruning steps, (f) quantification schedule using T' for the first image step and T'' for all following steps, (g) pruned schedule using T''' for the second and following steps, (h) over-approximated reachability result using pruned schedule (top part) and result after applying one correction step using T''

that the application of a transition relation which has been forward pruned once can only result in an *over-shooting* by at most one transition. In other words, in case of simple forward pruning, the maximum distance of an over-approximated state from a truly reachable state is one. In general, the application of j forward pruning iterations results in a maximum over-shooting distance of j. Therefore, the over-approximated reachability results can be corrected by applying a sequence of j additional image steps using the unpruned transition relation. The bottom part of Fig. 2(h) illustrates this correction for the given example.

In summary, the application of pruning requires that the image computation at step i must not use a transition relation that has been backward pruned more than $i-1$ times. Furthermore, over-approximated reachability results that are produced by a transition relation derived from j forward pruning iterations, can be corrected by j application of the exact transition relation. In Section 3.4 we describe an algorithm that dynamically prunes the transition relation as the state traversal progresses.

The advantage of the presented approach is that the majority of image computation steps can use the pruned transition relation. Once a fixpoint is reached, the exact set of reachable states is determined from the over-approximation by applying one or more exact steps. Since the computational bottleneck of BDD-based reachability analysis typically occurs in the middle of the traversal, this methods can significantly improve the overall efficiency. Further, the over-approximation is tight, in many practical cases even exact. If a given set of properties can be proven for this tight approximation, the correction step is not needed.

3 Algorithms

3.1 Preliminaries

We model a sequential circuit as a transition structure $K = \langle T, I \rangle$ consisting of a transition relation $T(y,x)$ and an initial predicate $I(x)$. The binary variables $x = \{x_1, \ldots, x_m\}$ are the current state variables, while the binary variables $y = \{y_1, \ldots, y_m\}$ are the next state variables. Given a predicate $P(x)$ describing a set of present states, the set of their successors, $S(y)$, is the *image* of $P(x)$ and is given by

$$S(y) = \exists x \,.\, P(x) \wedge T(y,x) \ . \tag{1}$$

The states of K reachable from the states in I can be computed by successive image computations. Denoting by $\mathsf{EY}\, P(x)$ the predicate obtained by replacing the y variables with the x variables in the image of $P(x)$, the reachable states are given by

$$R(x) = \mu Z(x) \,.\, I(x) \vee \mathsf{EY}\, Z(x) \ , \tag{2}$$

where μ indicates computation of the least fixpoint. We assume that the transition relation is given as the composition of elementary relations. If $w = \{w_1, \ldots, w_n\}$ is a set of binary *internal* variables with $n \geq m$, our assumption amounts to writing:

$$T(y,x) = \exists w \,.\, \bigwedge_{1 \leq i \leq m} (y_i \leftrightarrow w_i) \wedge \bigwedge_{1 \leq i \leq n} T_i(w,x) \ . \tag{3}$$

The variables in w are usually associated with the outputs of the combinational logic gates of the sequential circuit; each T_i is called a *gate relation* because it usually describes the behavior of a logic gate. For instance, if w_i is the output variable of a two-input AND gate with inputs w_j and x_k, then $T_i = w_i \leftrightarrow (w_j \wedge x_k)$. If, on the other hand, w_i is a primary input to the circuit, then $T_i = 1$. Each term of the form $y_i \leftrightarrow w_i$ equates a next state variable to an internal variable. (The output of the gate feeding the i-th memory element.)

When computing (2) for large circuits it is seldom practical to evaluate $T(y, x)$ before computing the images. This is especially true when BDDs [3] are used to represent the relations. Instead, one substitutes (3) into (1) to get

$$S(y) = \exists x \,.\, \exists w \,.\, P(x) \wedge \bigwedge_{1 \le i \le m} (y_i \leftrightarrow w_i) \wedge \bigwedge_{1 \le i \le n} T_i(w, x) \ . \tag{4}$$

The main advantage of using (4) stems from the ability to apply *early quantification* while conjoining the terms of the transition relation to the set of states. Indeed,

$$\exists a \,.\, g(b) \wedge f(a, b) = g(b) \wedge \exists a \,.\, f(a, b) \ , \tag{5}$$

though, in general, existential quantification does not distribute over conjunction. Application of (5) turns an image computation into a series of passes. Each pass replaces two terms with their conjunction and quantifies all variables that appear in only one of the resulting terms.

Early quantification reduces the number of variables that appear in the BDDs obtained as intermediate results during the computation of (4). Its impact depends largely on the order in which the terms are conjoined. The *conjunction scheduling problem* is the problem of determining an order of the terms in (4) that reduces the time and memory requirements of image computation. The evaluation of (2) requires in general repeated image computations. It is usually advantageous to take the conjunctions in (4) that do not involve $P(x)$ only once before reachability analysis is started. After these conjunctions have been taken, image computation amounts to evaluating

$$S(y) = \exists x \,.\, \exists w \,.\, P(x) \wedge \bigwedge_{1 \le i \le k} C_i(y, w, x) \ , \tag{6}$$

where each C_i is a *cluster* obtained by conjoining one or more terms from (4), and quantifying w variables not appearing in any other cluster. During the computation, early quantification is applied according to the following scheme.

$$S(y) = \exists v^1 \,.\, (C_1 \wedge \cdots \wedge \exists v^k \,.\, (C_k \wedge \exists v^{k+1} \,.\, P)) \ , \tag{7}$$

where v^i is the set of variables in $(x \cup w) \setminus \bigcup_{i<j\le k} v^j$ that do not appear in $C_1, \ldots, C_{i-1}$.

Too many clusters lead to needless recomputation, whereas too few clusters, or ill-assorted clusters, may adversely affect early quantification. The *clustering problem* is the problem of finding a suitable partition of the terms of (4) into clusters that reduces the time and memory requirements of image computation.

Though in principle the scheduling and clustering problems cannot be separated, it is common practice to order the conjuncts before clustering them. Clustering is then restricted to terms that form intervals in the order [17]. (See, however, [14] for a dissenting voice.)

3.2 Conjunction Scheduling via Linear Arrangement

We formulate the conjunction scheduling problem in terms of linear arrangement of a hypergraph.

Definition 1. *A* hypergraph $G = (V, H)$ *consists of a set of vertices* V *and a multiset of hyperedges* H*. Each hyperedge is a subset of* V*. A* linear arrangement *of* G *is a bijection* $\alpha : V \to \{1, \ldots, |V|\}$.

The maximum cut-width $\Gamma(G, \alpha)$ of hypergraph G under linear arrangement α is the maximum number of hyperedges crossing a section of the graph. Formally,

$$\Gamma(G, \alpha) = \max_{1 \le i \le |V|} |\{h \in H : \exists u \in h \,.\, \exists v \in h \,.\, \alpha(u) \le i \le \alpha(v)\}| \ . \tag{8}$$

The *minimum max-cut problem* asks for a linear arrangement α of hypergraph G that minimizes $\Gamma(G, \alpha)$. This problem is NP-complete [11], but effective heuristic techniques have been developed for it.

Given a set of clusters $C = \{C_i(y, w, x)\}$, let $\sigma(C)$ be the set of variables appearing in the clusters of C. We consider the hypergraph

$$G_C = (C, \{\{C_i : v \in \sigma(\{C_i\})\} : v \in y \cup w \cup x\}) \ . \tag{9}$$

In G_C each vertex models a cluster, and each hyperedge models a variable. The vertices connected by a hyperedge are the clusters in which the corresponding variable appears. Hyperedges may be repeated because several variables may appear in exactly the same clusters. (This is a slight departure from the standard definition of hypergraphs.)

A linear arrangement α of G_C corresponds to a conjunction schedule for image computation. The maximum cut-width $\Gamma(G_C, \alpha)$ is the maximum number of live variables during image computation. Solving the minimum max-cut problem for G_C is therefore related to finding a good conjunction schedule.

When the transition relation of structure K is given in the form of a logic circuit, the list of clusters C is obtained as follows. Initially, one conjunct is created for each gate in the circuit, and for each next state function as shown in (3). For large circuits, the resulting large number of conjuncts may hinder the computation of a good linear arrangement. Therefore, *fanout-free* regions of the circuit are selectively collapsed.

The same conjunction schedule for a given transition relation is typically used for the computation of the images of several sets of states. In general, the predicates representing these states will depend on different sets of variables. We make the conservative assumption that all current state variables appear in these sets of states. This is achieved by augmenting C with a dummy cluster that depends on all variables in x. The position of this cluster is fixed at the beginning of the linear arrangement. Likewise, a second dummy cluster depending on all next state variables is added to C and its position is fixed at the end of the linear arrangement.

Once the initial list of clusters C is obtained, we invoke CAPO, [7] to obtain a single-row placement of the vertices of G_C. CAPO produces a linear arrangement with a small maximum cut-width, while also trying to reduce the total wire length. The length of a (non-empty) hyperedge is the maximum distance in the arrangement between two

vertices belonging to the hyperedge. In the context of image computation, the maximum cut-width translates into the peak number of variables during image computation, while the length of a hyperedge corresponds to the lifetime of a variable. Is is noted in [15] that reducing the average lifetime of variables reduces the sizes of the BDDs and the cost of operating on them. In terms of the *dependency matrix* of the transition relation, a good linear arrangement results in a small *bandwidth*.

It is also possible to use linear arrangement to produce a static variable order for the BDDs (cf. [13]). For that we model a variable as a vertex, and a cluster as a hyperedge connecting all the variables appearing in the cluster. The BDDs of the clusters are ordered according to the arrangement of the vertices before the clustering algorithm of Section 3.3 is applied.

3.3 Clustering

Clustering reduces the number of conjunctions that must be taken during image computation by collapsing groups of clusters. Another important objective is to make the early quantification of variables from the transition relation possible.

The input to clustering is a linearly arranged set of clusters. The output is a reduced set of clusters. Each output cluster is the conjunction of a set of contiguous input clusters. In other words, the clustering process respects the given linear arrangement.

Definition 2. *Given a linear arrangement α of a set of clusters C, a variable v is* dead *outside positions i and j if v appears in cluster C_k only if $i \leq \alpha(C_k) \leq j$. The number of variables that are dead outside positions i and j is denoted by $D_{i,j}$. The number of variables that are not dead outside positions i and j, and that appear in clusters C_k such that $i \leq \alpha(C_k) \leq j$ is denoted by $V_{i,j}$.*

According to this definition, $D_{i,j}$ is the number of variables that can be quantified if the clusters between positions i and j are merged, while $V_{i,j}$ is an upper bound on the number of variables appearing in the result of the merger after quantification.

Fig. 3 shows a heuristic clustering algorithm. The algorithm iterates until no new clusters are created in one pass. At each pass, it creates a list of candidates. Each candidate is a contiguous set of clusters. The list is ordered in decreasing order of $D_{i,j}$ to favor candidates that allow many variables to be quantified. As a tie-breaker, the upper bound on the number of variables in the resulting cluster is used. This policy favors the creation of small clusters that may be merged in subsequent passes.

The number of candidates considered by the algorithm of Fig. 3 is quadratic in the number of clusters. This may be inefficient. Therefore, the actual implementation limits the maximum number of clusters in a candidate to 200.

Once the list has been sorted, the candidates are examined in turn. If the result of merging all the clusters in the candidate is smaller than a specified threshold, the candidate is accepted and the result of the merger replaces all the clusters in the candidate. All other candidates having clusters in common with the accepted one are rejected.

To limit the cost of this phase, the conjunction of the clusters in a candidate is abandoned as soon as it exceeds the threshold (even though conjoining more clusters may eventually bring the size down again). Furthermore, all candidates that are supersets

```
GREEDYCLUSTERING(C) {
    while (1) {
        Calculate D_{i,j} and V_{i,j} of C;
        nClusteringDone = 0;
        F = ();
        for (i = 0; i < |C|; i++) {
            used[i] = new[i] = false;
            for (j = i + 1; j < |C|; j++)
                Insert quadruple f = (i, j, D_{i,j}, V_{i,j}) into F;
        }
        Sort F in order of decreasing D_{i,j}
          in case of tie, give higher priority to entries with smaller V_{i,j};
        for each f in F {
            if (used[i] || ··· || used[j]) continue ;
            C_{i,j} = C_i ∧ ··· ∧ C_j;
            if (BDDSIZE(C_{i,j}) <ThesholdValue) {
                nClusteringDone++;
                for (k = i; k < j; k++) used[k] = true;
                C_j = C_{i,j};
                new[j] = true;
            } else free C_{i,j};
        }
        if (nClusteringDone == 0) break ;
        Remove from C clusters that are marked used and not new;
    }
}
```

Fig. 3. Greedy clustering algorithm

of the set of clusters whose conjunction exceeded the threshold are discarded. These details are omitted from the pseudocode of Fig. 3 to avoid clutter.

3.4 Pruning

During the initial linear arrangement and clustering the assumption is made that all current state variables will appear in the predicate $P(x)$ whose image must be computed. As seen in Section 2, relaxing this assumption may lead to more extensive application of early quantification. We now describe how this process is carried out.

The algorithm of Fig. 4 applies pruning while computing the states reachable from I according to the transition relation described by C.

Theorem 1. *If forward pruning is excluded, then algorithm* REACHABLE *of Fig. 4 correctly computes the states reachable from I according to C.*

Proof. We show by induction that at the i-th iterations of the main loop $R(x)$ describes the states reachable from I in i steps or less. This trivially holds for $i = 0$. For the

```
REACHABLE (C, I) {
    R(x) = P(x) = I(x);
    while (1) {
        PRUNE (C, P);
        S(x) = EY P(x);
        N(x) = S(x) ∧ ¬R(x);
        if (N(x) = 0) return R(x);
        R(x) = R(x) ∨ N(x);
        P(x) = PICK (N(x), S(x));  // choose a small BDD for P such that N ≤ P ≤ S
        existentially quantify from P(x) the variables not in S(x);
    }
}

PRUNE (C, P) {
    X = {x_j : y_j ∉ σ(C) ∧ x_j ∉ σ({P}) ∧ ∃i . x_j ∈ σ({C_i}) ∧ x_j ∉ σ(C \ {C_i})};
    C = ∃X . C;  // backward pruning
    Y = {y_j : x_j ∉ σ(C) ∧ ∃i . y_j ∈ σ({C_i}) ∧ y_j ∉ σ(C \ {C_i})};
    C = ∃Y . C;  // forward pruning
    do { // intermediate variable quantification
        W = {w_j : ∃i . w_j ∈ σ({C_i}) ∧ w_j ∉ σ(C \ {C_i})};
        C = ∃W . C;
    } while (W ≠ ∅);
}
```

Fig. 4. Reachability analysis algorithm

inductive step, we prove that pruning the transition relation does not change the result of successive image computations. Since pruning consists of existential quantifications, the new transition relation contains the old one. Hence, no state is dropped from $S(x)$.

To see that no states are added, assume that at the start of the i-th iteration, P describes states reachable from I in exactly $i-1$ steps, and the states reachable in i or more steps can be correctly computed using the current C.

Suppose x_j is pruned at the i-th iteration. Then, it does not appear in P, and it appears in exactly one C_i. Therefore, if the original C were used for image computation, early quantification would apply, and pruning of x_j would occur as part of image computation. Hence, pruning of x_j does not affect S. By simple induction on the number of pruned variables one concludes that backward pruning does not change the result of image computation. (Quantifying w_j obviously does not change the result if w_j appears in exactly one C_i.)

Since y_j does not appear in C if x_j is pruned, then x_j does not appear in $S(x)$. Furthermore, the choice of P for the next iteration preserves this property. Since $P \leq S$, after quantification of the variables not in S from P, the result is still contained in S. Since the variables not in the incoming P are not in the new P either, the pruning at iteration i is valid also at successive iterations. □

Notice that the quantification of the variables not in S from P is can be avoided at the expense of complicating the proof. Moreover, the *restrict* operator [10] can be used to select P so that no variables not in S appears in P.

It should be noted that the effectiveness of pruning is enhanced by the ability of the clustering algorithm to reduce the number of terms in which a variable appears.

Lemma 1. *Let $T_0, \ldots, T_k$ the sequence of transition relations generated by forward pruning. Specifically, let $T_0 = T$, and $T_{i+1} = \exists Y_i \,.\, T_i$, where Y_i is the set of next-state variables quantified because their corresponding current-state variables X_i are not in T_i. (That is, $X_i \cap \sigma(T_i) = \emptyset$ and $Y_i \cap Y_j = \emptyset$ for $0 \leq j < i$.) Let EY_i compute the successors of a set of states using transition relation T_i. Then*

$$\exists X_i \,.\, \mathsf{EY}_i \, Z(x) = \mathsf{EY}_{i+1} \, Z(x) \ .$$

Lemma 2. $R_{i+1}(x) \vee \exists X_i \,.\, I(x) = \exists X_i \,.\, R_i(x)$.

Proof. Expanding both sides, and observing that $I(x) \leq \exists X_i \,.\, I(x)$, we get

$$\exists X_i \,.\, I(x) \vee \bigvee_{j>0} \mathsf{EY}_{i+1}^j \, I(x) = \exists X_i \,.\, I(x) \vee \bigvee_{j>0} \exists X_i \,.\, \mathsf{EY}_i^j \, I(x) \ .$$

The two sides can be shown to be identical by recursive application of Lemma 1. □

Theorem 2. *Let $R^+(x)$ be the result produced by reachability with forward pruning. Then the reachable states are given by*

$$R(x) = \nu Z(x) \,.\, R^+(x) \wedge (I(x) \vee \mathsf{EY} \, Z(x)) \ . \tag{10}$$

Proof. Let $R_i(x)$ be the result of reachability analysis using T_i. We have:

$$R_i(x) = I(x) \vee \mathsf{EY}_i \, R_i(x) \ ,$$

which, applying (5) and Lemma 2, becomes

$$R_i(x) = I(x) \vee \mathsf{EY}_i \, R_{i+1}(x) \ . \tag{11}$$

Therefore, we can compute the exact reachable states $R(x) = R_0(x)$ by starting with $R^+(x) = R_k(x)$, and iteratively applying (11). We now observe that $T_i \geq T$ for $0 \leq i \leq k$, and consequently,

$$R_0(x) \leq I(x) \vee \mathsf{EY} \, R_{i+1}(x) \leq R_i(x) \ .$$

Therefore, $R_0(x) = \nu Z(x) \,.\, R^+(x) \wedge (I(x) \vee \mathsf{EY} \, Z(x))$. Convergence is guaranteed in k iterations. □

Theorem 3. *Procedure* REACHABLE *computes the reachable states of $K = \langle T, I \rangle$.*

Proof (sketch). To account for the interleaving of backward and forward pruning, we observe that each transition relation T_i obtained by forward pruning is applied from a set of set of states that is between $I(x)$ and $R_{i-1}(x) \leq R_i(x)$. Hence, if used to convergence, it computes $R_i(x)$. □

4 Experiments

We implemented the proposed method in VIS [2]. Experiments are conducted on 1.7GHz Pentium 4 with 1GB of RAM running Linux. We compare the fine-grain method (denoted by FG) to the IWLS95 [17], Hybrid [16], and MLP [15] methods. In these experiments, we turned on dynamic variable ordering for BDDs and we set the data size limit to 700MB. Figure 5 compares the four methods in the context of reachability analysis. Each experiment was allotted 20,000 seconds.

The experiments show overall improvements in CPU time and memory usage. The proposed method outperforms the IWLS95, Hybrid, and MLP methods in most hard benchmark examples, such as s4863, am2901, prolog, and s3330. In the case of rotate32, which is the 32-bit rotator, all the transition functions contain all the present state variables. It means that there is no good quantification schedule without intermediate variables. The good result for FG witnesses its ability to choose a good set of intermediate variables.

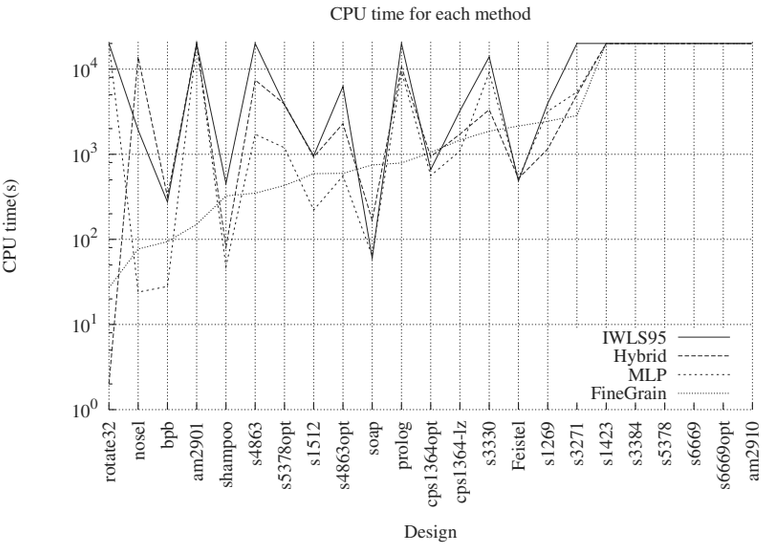


Fig. 5. Performance comparison

Table 1 reports the number of state variables before and after applying pruning. We can prune all the state variables in the case of s4863 and s4863opt; hence, the transition relations can be reduced to the constant 1. This means that the reachability analysis step is trivial. However, the recovery step is hard because it computes the exact reachable states from the set of all states. In Fig. 5, most CPU time of s4863 and s4863opt is consumed by the recovery step.

Table 1. Number of state variables before and after optimization

	Before		After	
Design	Present State	Next State	Present State	Next State
prolog	136	136	114	112
s1269	37	37	36	36
s1423	74	74	72	72
s1512	57	57	46	46
s3271	116	116	115	115
s3330	132	132	113	112
s3384	183	183	120	119
s4863	104	104	0	0
s4863opt	88	88	0	5
s5378	179	179	159	159
s5378opt	121	121	83	83
s6669	239	239	175	164
s6669opt	231	231	167	148

5 Conclusions

We have presented an approach to image computation for symbolic reachability analysis that exploits the fine-grain structure of the transition relation. In the case of hardware circuits, such structure is represented by the combinational gates that make up the next-state functions; the approach, however, is general. The advantage of a fine-grain approach is the ability to accurately place intermediate variables in the transition relation so as to promote extensive early quantification. We have presented three techniques that rely on this feature: A procedure for the computation of the conjunction schedule based on minimum max-cut linear arrangement; a clustering algorithm, and a variable pruning algorithm that works in conjunction with the standard symbolic reachability analysis procedure.

Preliminary experimental results show great promise for the fine-grain approach, especially in cases when the circuit implementing the transition relations are deep and have large BDDs. In these cases other image computation procedures tend to place intermediate variables suboptimally. There is still considerable work to be done to reduce the overhead of our clustering algorithm.

Acknowledgments. The authors would like to thank Igor Markov of the University of Michigan for his help in installing and customizing CAPO for the experiments.

References

[1] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Fifth International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, pages 193–207, Amsterdam, The Netherlands, March 1999.

[2] R. K. Brayton et al. VIS: A system for verification and synthesis. In T. Henzinger and R. Alur, editors, *Eighth Conference on Computer Aided Verification (CAV'96)*, pages 428–432. Springer-Verlag, Rutgers University, 1996. LNCS 1102.

[3] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[4] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design*, 13(4):401–424, April 1994.

[5] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, pages 428–439, June 1990.

[6] G. Cabodi, P. Camurati, and S. Quer. Improved reachability analysis of large finite state machines. In *Proceedings of the International Conference on Computer-Aided Design*, pages 354–360, Santa Clara, CA, November 1996.

[7] A. E. Caldwell, A. B. Kahng, and I. Markov. Optimal partitioners and end-case placers for standard-cell layout. *IEEE Transactions on Computer-Aided Design*, 19(11):1304–1314, November 2000.

[8] P. P. Chauhan, E. M. Clarke, S. Jha, J. Kukula, T. Shiple, H. Veith, and D. Wang. Non-linear quantification scheduling in image computation. In *Proceedings of the International Conference on Computer-Aided Design*, pages 293–298, San Jose, CA, November 2001.

[9] O. Coudert, C. Berthet, and J. C. Madre. Verification of sequential machines based on symbolic execution. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, pages 365–373. Springer-Verlag, 1989. LNCS 407.

[10] O. Coudert and J. C. Madre. A unified framework for the formal verification of sequential circuits. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 126–129, November 1990.

[11] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, pages 237–267, 1987.

[12] D. Geist and I. Beer. Efficient model checking by automated ordering of transition relation partitions. In D. L. Dill, editor, *Sixth Conference on Computer Aided Verification (CAV'94)*, pages 299–310, Berlin, 1994. Springer-Verlag. LNCS 818.

[13] A. Gupta, Z. Yang, L. Zhang, and S. Malik. Partition-based decision heuristics for image computation using SAT and BDDs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 286–292, San Jose, CA, November 2001.

[14] R. Hojati, S. C. Krishnan, and R. K. Brayton. Early quantification and partitioned transition relations. In *Proceedings of the International Conference on Computer Design*, pages 12–19, Austin, TX, October 1996.

[15] I-H. Moon, G. D. Hachtel, and F. Somenzi. Border-block triangular form and conjunction schedule in image computation. In W. A. Hunt, Jr. and S. D. Johnson, editors, *Formal Methods in Computer Aided Design*, pages 73–90. Springer-Verlag, November 2000. LNCS 1954.

[16] I.-H. Moon, J. H. Kukula, K. Ravi, and F. Somenzi. To split or to conjoin: The question in image computation. In *Proceedings of the Design Automation Conference*, pages 23–28, Los Angeles, CA, June 2000.

[17] R. K. Ranjan, A. Aziz, R. K. Brayton, B. F. Plessier, and C. Pixley. Efficient BDD algorithms for FSM synthesis and verification. Presented at IWLS95, Lake Tahoe, CA., May 1995.

[18] C. H. West and P. Zafiropulo. Automated validation of a communications protocol: The CCITT X.21 recommendation. *IBM Journal of Research and Development*, 22:60–71, 1978.