Non-linear Quantification Scheduling in Image Computation

Pankaj Chauhan¹, Edmund M. Clarke¹, Somesh Jha², Jim Kukula³, Tom Shiple³, Helmut Veith³, Dong Wang³

¹ Carnegie Mellon University, Pittsburgh, PA ² University of Wisconsin, Madison, WI ³ TU Vienna, Austria

ABSTRACT
Computing the set of states reachable in one step from a given set of states, i.e., image computation, is a crucial step in several symbolic verification algorithms, including model checking and reachability analysis. So far, the best methods for quantification scheduling in image computation, with a conjunctively partitioned transition relation, have been restricted to a linear schedule. This results in a loss of flexibility during image computation. We view image computation as a problem of constructing an optimal parse tree for the image set. The optimality of a parse tree is defined by the largest BDD that is encountered during the computation of the tree. We present dynamic and static versions of a new algorithm, VarScore, which exploits the flexibility offered by the parse tree approach to the image computation. We show by extensive experimentation that our techniques outperform the best known techniques so far.

1. INTRODUCTION
Symbolic representation of transition relations and state sets using Binary Decision Diagrams or BDDs [3, 9, 13] has led to a breakthrough in the use of model checking and reachability analysis. The transition relation $R(s, w, s')$, where $s$ and $s'$ are present and next states respectively and $w$ are inputs, is represented by the characteristic function of the set of transitions that comprise $R$. Similarly, state sets are also represented using characteristic functions of the sets.

At the core of all symbolic algorithms is image computation¹, i.e., the task of computing the set of successors $\text{Img}(S)$ of a set of states $S$, where

$$\text{Img}(S) := \{s' : \exists w. R(s, w, s') \land s \in S\}.$$  

Image computation is one of the major bottlenecks in verification. Often it is impossible to construct a single BDD for the transition relation $R$. Instead, $R$ is represented as a partitioned transition relation, i.e., as the conjunction of several BDDs, each representing a part of $R$. The problem is to compute $\text{Img}(S)$ without actually computing $R$.

The definition of $\text{Img}$ involves evaluation of a quantified Boolean formula. In the BDD representation, this amounts to quantifying over several Boolean state variables. Early quantification [5, 17] is based on the following Boolean equation:

$$\exists y. f(x, y) \land g(x) \equiv (\exists y f(x, y)) \land g(x) \tag{1}$$

Early quantification results in smaller intermediate BDDs by reducing the scope of each variable to be quantified. The success of early quantification heavily depends upon the derivation and ordering of the sub-relations which comprise $R$. This problem has attracted significant attention over the last decade. Since the problem is known to be NP-hard [12], various heuristics have been proposed for the problem.

This paper offers a more flexible approach to image computation by viewing the image computation equation as a symbolic expression evaluation problem. The main contributions of this paper are as follows:

- We formulate the problem of image computation as an expression evaluation problem where the goal is to reduce the size of the intermediate BDDs as in [12]. This approach provides significantly more flexibility than the traditional linear approach for ordering BDDs during image computation. We show how this approach subsumes the linear approaches.

- We provide the VarScore heuristics for evaluating the parse tree of image computation equation to reduce the size of the intermediate BDDs. Our heuristics are based on scoring the variables that need to be quantified and restructuring the parse tree according to the heuristic. We provide dynamic and static versions of our VarScore heuristics. In the dynamic version, the parse tree is built for each image computation, while in the static version, a single parse tree is built in the beginning and is used for all subsequent image computations.

- We compare our dynamic and static heuristics to the best known techniques based on linear ordering of BDDs. We show that even with a simple heuristic such as VarScore, we achieve impressive results. We have also contributed to the...
code base of the symbolic model checker NuSMV [8] by implementing our techniques.

The rest of the paper is organized as follows: In Section 2, we introduce notations and definitions and review the current state of the art for this problem. Section 3 describes our basic approach and heuristics. Section 4 describes experimental results. Finally, we conclude in Section 5 with some directions for future research.

2. PRELIMINARIES AND RELATED WORK

Notation: Every state is represented as a vector \( b_1 \ldots b_n \in \{0,1\}^n \) of Boolean values. The transition relation \( R \) is represented by a Boolean function \( T(x_1, \ldots, x_n, w_1, \ldots, w_m, x'_1, \ldots, x'_n) \).

Variables \( x = x_1, \ldots, x_n \), \( X' = x'_1, \ldots, x'_n \) and \( W = w_1, \ldots, w_m \) are current state, next state and input variables respectively. \( T(x_1, \ldots, x_n, w_1, \ldots, w_m, x'_1, \ldots, x'_n) \) is abbreviated as \( T(X, W, X') \). Similarly, functions of the form \( S(X) = S(x_1, \ldots, x_n) \) describe sets of states. The set of variables on which \( f \) depends on is denoted by \( \text{Supp}(f) \).

Example 1. [3 bit counter. (Running Example)] Consider a 3-bit counter with bits \( x_1, x_2 \) and \( x_3 \), where \( x_1 \) is the least significant and \( x_3 \) the most significant bit. The state variables are \( X = x_1, x_2, x_3, X' = x'_1, x'_2, x'_3 \). The transition relation of the counter can be expressed as

\[
T(X, W, X') = (x'_1 \leftrightarrow \neg x_1) \land (x'_2 \leftrightarrow x_1 \oplus x_2) \land (x'_3 \leftrightarrow x_1 \land x_2 \oplus x_3).
\]

Note that the counter does not have any input variables. In later examples, we will compute the image \( \text{Img}(S) \) of the set \( S(X) = \neg x_1 \) which contains those states where the counter is even.

Partitioned BDDs: For most realistic designs it is impossible to build a single BDD for the entire transition relation. Therefore, it is common to represent the transition relation as a conjunction of smaller BDDs \( T_1(X, W, X'), T_2(X, W, X'), \ldots, T_i(X, W, X') \), i.e.,

\[
T(X, W, X') = \bigwedge_{1 \leq i \leq l} T_i(X, W, X'),
\]

where each \( T_i \) is represented as a BDD. The sequence \( T_1, \ldots, T_l \) is called a conjunctively partitioned transition relation. Note that \( T \) is not actually computed, and only the \( T_i \)'s are kept in memory. Typically, these partitions are derived from the next state functions of state variables. However, if the BDD of a single next state function is too large, then the circuit for the next state function is further partitioned by introducing cut-point variables \( C = c_1, \ldots, c_p \). These cut-points are then quantified away in the image computation. Let \( Q \) denote the variables to be quantified, which in our case is \( Q = X \cup W \cup C \) and \( |Q| = n + m + p \). The equation for image computation is then:

\[
\text{Img}(S(X)) = \exists Q. (T(X, W, C, X') \land S(X)) \tag{2}
\]

\[
= \exists Q. (\bigwedge_{1 \leq i \leq l} T_i(X, W, C, X') \land S(X)) \tag{3}
\]

Example 2. [3 bit counter, etc.] For the 3 bit counter, a very simple partitioned transition relation is given by the functions \( T_1 = (x'_1 \leftrightarrow \neg x_1), T_2 = (x'_2 \leftrightarrow x_1 \oplus x_2) \) and \( T_3 = (x'_3 \leftrightarrow (x_1 \land x_2) \oplus x_3) \).

Early Quantification: Usually, the size of a BDD reduces by quantifying away a variable in its support. Loosely speaking, BDDs in the partition correspond to semantic entities of the design to be verified and it is expected that not all variables appear in all clusters.

Therefore, by virtue of Equation 1, some of the quantifications in Equation 3 may be shifted over several BDDs as follows:

\[
\text{Img}(S(X)) = \exists Q_1 \cdot (T_1 \land \exists Q_2 \cdot (T_2 \ldots \exists Q_l \cdot (T_l \land S(X)))) \tag{4}
\]

where \( Q_i \) is the set of variables which do not appear in \( \text{Supp}(T_1) \cup \ldots \cup \text{Supp}(T_{l-1}) \). If we look at the parse tree of this equation, we see that it is a linear chain of conjunctions and quantifications. Generalizing this for an arbitrary parse tree, a variable can be quantified away at a sub-tree node as soon as it does not appear in the rest of the tree.

Quantification Scheduling: The size of intermediate BDDs and effectiveness of early quantification depends heavily upon the order in which BDDs are conjoined in Equation 4. For each linear ordering of the conjunctions, there is a unique order of variable quantifications. The problem of ordering the BDDs so as to minimize the size of intermediate BDDs is known as the quantification scheduling problem. The order of BDDs is known as the conjunction schedule. Traditionally, only linear conjunction schedules have been considered. We generalize this concept to arbitrary parse trees of the image computation equation. The problem of building the parse tree and scheduling the quantifications over them is called the quantification scheduling problem.

Related Work: Burch et al. [4] and Touati et al. [17] first recognized the importance of early quantification for image computation. Geist and Beer [10] proposed a simple heuristic algorithm, in which they ordered conjuncts in the increasing order of the number of support variables in the conjunct. Hojati et al. were the first to formulate the early quantification problem as an evaluation of a parse tree and proved the NP-completeness of the problem. They also offered a greedy strategy for evaluation of the parse tree by evaluating the node with the smallest support set next. However, they did not compare their technique against other techniques, so the effectiveness of their algorithms was unclear. Traditional techniques for linear quantification schedules begin by first ordering the conjuncts, and then clustering them, and finally ordering the clusters again using the same heuristics. Ranjan et al. [16] proposed the first successful heuristics for this problem and Yang [18] refined their technique. Their ordering procedure linearly orders the BDDs based on a heuristic score. The individual BDDs are then formed into clusters by conjoining them according to the linear order until BDD size grows beyond certain threshold. Finally, these clusters are ordered using the same algorithm. A recent paper by Moon and Somenzi [15] presents an ordering algorithm (henceforth referred to as FMCAD00) based on computing the border block triangular form of the dependence matrix to minimize the average active lifetime of variables. Their clustering algorithm is based on the sharing of support variables or affinity between conjuncts. They extended their notion of lifetimes and used combinatorial algorithms to improve the performance [7].

Research has also been carried out in disjunctive decomposition of transition relation. In [14], the authors use dependency matrix to decide whether to introduce disjunctive decomposition. However, after the decomposition they use standard linear quantification schedules. Gupta et al. [11] use SAT procedures to derive finer disjunctive decomposition of the transition relation and use a conjunctive schedule for the subproblems corresponding to the clauses. They also propose to use a non-linear quantification scheduling algorithm similar to the one proposed in this paper for leaf image computation. They also use variable scoring mechanism to choose a variable. Then all BDD relations that the variable appears in are conjoined along with quantification of the chosen variable. However, they presented experimental results in the context of using...
SAT for decomposition of the overall problem, and it is difficult to make a fair comparison against a purely BDD based approach. Apart from this, there are some differences in the details of the algorithm. One difference is in the scoring mechanism. They use the product of the BDD sizes as the heuristic score, while we use the sum or the sum of the squares of the BDD sizes. In the worst case, it is true that the size of the result of the apply operation will the the product of the sizes of two BDDs, but this is a rather pessimistic estimate. Better estimation of BDD sizes as a function of the support set will improve the heuristics. We also believe that our algorithms provides more flexibility by conjoining only two smallest BDDs for a variable, unlike algorithm where they conjoin all the BDDs (which is used in [11]).

3. **VARSCORE ALGORITHMS**

In this section, we describe the VarScore heuristic algorithms for the quantification scheduling problem. First, we describe the dynamic version of VarScore algorithm, where a parse tree is built for each image computation. Next, we describe static versions of VarScore algorithm, where the parse tree is built only once and used for all subsequent image computations. In static versions of our algorithm the information about the state set \( S(X) \) is not available (see Equation 3). Therefore, the heuristic scores are approximations. The basic step of our algorithms is described in Figure 1.

![Figure 1: Basic step of the VarScore algorithms](image)

The input to **VARSCOREBASICSTEP** is a set of variables \( Q \) to be quantified, and a collection \( F \) of BDDs. First, any variable that appears in the support of only one BDD is immediately quantified away and the sets \( F \) and \( Q \) are adjusted accordingly (lines 1–3). Otherwise a heuristic score is computed for the variables in \( Q \). The variable with the lowest score, say \( q \), is chosen next and the two smallest BDDs in whose support that variable appears are conjoined. For efficiency reasons, if \( q \) appears in the support of only those two BDDs, then we use **BDDAndExists** operation to conjoin and quantify away that variable.

In the **dynamic version** of the algorithm, this step is called repeatedly for each image computation, beginning with \( F = \{ T_1, \ldots, T_i, S \} \) and \( Q = X \cup W \cup C \). \( F \) can also be seen as a collection of parse subtrees (or forest) where the BDD operations are carried out at the roots of the subtrees. When all the variables are quantified (\( Q = \emptyset \)), remaining BDDs from \( F \) are conjoined in any arbitrary order to compute \( \text{Img}(S) \). The scoring algorithm that we use is very simple:

we sum up the sizes of the BDDs in which a particular variable appears.

However, we are also investigating other more complex scoring algorithms. Figure 2 illustrates the dynamic algorithm on our 3-bit counter example.
Second Static Approach: Note that in the first static approach, we cannot quantify away any present state variable as we do not have information about $S$. Thus the parse tree that is built in the beginning does not take into account the present state variables and $S$. However, introducing $S$ as early in the Equation 3 restricts the conjunct BDDs, often reducing their sizes [11, 14]. Moreover, the bulk of the variables affecting the computation are these present state variables. In fact, if we remove the BDD size constraint, we end up with a monolithic representation of the transition relation! To alleviate this problem, we propose a second static approach that takes into effect the present state variables. We build an approximation of the parse tree but not the tree itself. Instead of working with the actual BDDs, we work only with the support sets of BDDs. The size of a BDD $T_i$ is estimated to be some function of $|\text{Supp}(T_i)|$. The linear function $\text{size}(T_i) = |\text{Supp}(T_i)|$ is an optimistic choice, while the exponential function $\text{size}(T_i) = 2^{|\text{Supp}(T_i)|}$ is too pessimistic. We have determined experimentally that a quadratic function $\text{size}(T_i) = 2^{|\text{Supp}(T_i)|^2}$ is a good estimate. Let $V$ be the set of boolean variables. Therefore, the support set of a boolean function $T_i$ is a subset of $V$ or is in the powerset of $V$ (denoted by $2^V$). Any function $f: 2^V \rightarrow N$ (where $N$ is the set of natural numbers) can be used in this approach. Intuitively, $f(V')$ approximates the size of the BDD of a boolean function with the support set $V'$. The following identities are used for adjusting the support sets after conjunction/quantification.

\[
\text{Supp}(\exists q.T_i) = \text{Supp}(T_i) \setminus \{q\}
\]
\[
\text{Supp}(T_i \land T_j) = \text{Supp}(T_i) \cup \text{Supp}(T_j)
\]

So we build the pseudo-parse tree with these approximations by calling VarScoreBasicStep repeatedly until $Q = \emptyset$. The remaining subtrees in $F$ are conjoined in arbitrary order to get a single parse tree. Here $F$ will denote the forest of the subtrees. We assume that $\text{Supp}(S) = X$. After building this pseudo-parse tree, we can see that all the subtrees not in the path from $S$ to the root can be evaluated in the beginning itself. Moreover, we do not need to take into account the BDD size constraint, because those same BDDs will have to be evaluated anyway. So we evaluate the remaining subtrees and get a linear chain from $S$ to the root. The quantifications for $X$ variables are scheduled anew for each image computation.

Third Static Approach: This approach is similar to the second static approach, but instead of working with the support sets, we work with actual BDDs. This provides a more accurate estimate of the sizes (compared to approximating the sizes of BDDs as some function of the size of support set). The BDD for $S$ is taken to be some reasonably complex BDD resembling the state set, e.g. the BDD for initial states or a random BDD with almost all $X$ variables in support. This is the only approximation introduced. The tree is built statically and all the subtrees not in the path from $S$ to the root are evaluated in the beginning. The quantifications along the path from $S$ to the root are scheduled for each image computation using the actual $S$, as in the second approach.

4. EXPERIMENTAL RESULTS

In order to evaluate the effectiveness of our algorithms, we ran reachability and model checking experiments on circuits obtained from the public domain and industry. The “S” series of circuits are ISCAS’93 benchmarks, and the “IU” series of circuits are various abstractions of an interface control circuit from Synopsys. For a fair comparison, we implemented all the techniques in the NuSMV model checker. All experiments were performed on a 200MHz quad Pentium Pro processor machine running the Linux operating system with 1GB of main memory. We restricted the memory usage to 900MB, but did not set a time limit. The two performance metrics we measured are running time and peak number of live BDD nodes. We provided a prior ordering to the model checker and turned off the dynamic variable reordering option. This was done so that the effects of BDD variable reordering do not “pollute” the result. We also recorded the amount of time spent before any image computation is done. The cost of this phase is amortized over several image computations performed during model checking and reachability analysis. In Table 1, we compare the three static techniques presented in this paper with FMCAD00 [15] and simulated annealing based techniques that appeared in [7]. The first column shows the name of the Circuit. The second column (marked as #FF) shows the number of state variables in the circuit. The next two columns (marked as #inp and log$_2$#reach) show the number of inputs and log of the number of reachable states in the various circuits. Subsequent columns show the performance results. Columns marked as FMCAD and SA refer to the algorithm presented in [15] and [7] respectively. The results corresponding to the three static strategies appear in the columns marked as VS-I, VS-II, and VS-III respectively.

We observe that VarScore algorithms is better in most of the cases against best of the simulated annealing and FMCAD00 methods. The margin of improvement is more in space than for time. Also observe that we spend significantly more time in the initial ordering phase (some time about 20% of the total time). Thus we have very good results when the number of image computations to be done are large, so that the cost of the initial phase is amortized. The average time speedup we observe is about 20% over the best of FMCAD00 and simulated annealing, while space savings are even better, about 40% for VS-III and about 20-30% on average for VS-I and VS-II.

5. CONCLUSIONS AND FUTURE WORK

We have proposed simple yet effective quantification scheduling algorithms for the image computation problem. We view the problem of quantification scheduling for symbolic image computation as a quantified Boolean formula evaluation problem. We have also proposed heuristic algorithms based on scoring of quantification variables to reduce the size of the intermediate BDDs. We have demonstrated that our simple yet flexible approach yields better experimental results for many reachability analysis and model checking problems.

There are a number of directions for future research. We can view the problem of building an optimal parse tree as a combinatorial optimization problem and apply techniques like simulated annealing to get better quantification schedules. A middle ground between dynamic and static techniques seems promising. For example, we believe that beginning with some static schedule, the schedule can be tuned for a particular image computation with a little effort. We also want to investigate techniques of approximating sizes of BDDs based on the size of support sets, which will definitely improve all image computation heuristics. Additional experiments are required to understand the relative performance of heuristics. We would also like to apply the techniques developed for quantification scheduling to other related problems, like splitting orders in SAT checkers [2] and hierarchical model checking [1].

6. REFERENCES

<table>
<thead>
<tr>
<th>Circuit</th>
<th>#F</th>
<th>#inp.</th>
<th>log of #reach</th>
<th>Total Time (secs)</th>
<th>Peak Live BDD Nodes (K)</th>
<th>Static phase time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLE</td>
<td>73</td>
<td>0</td>
<td>14.63</td>
<td>159 182 37 86 73</td>
<td>289 223 225 29 19</td>
<td>2 29 19 17 19</td>
</tr>
<tr>
<td>GUID</td>
<td>91</td>
<td>0</td>
<td>47.58</td>
<td>14 24 54 18 35</td>
<td>137 138 14 95 95</td>
<td>4 19 28 24 27</td>
</tr>
<tr>
<td>S953</td>
<td>29</td>
<td>16</td>
<td>8.98</td>
<td>1 3 4 2 2</td>
<td>15 15 14 19 16</td>
<td>1 3 2 1 1</td>
</tr>
<tr>
<td>IU30</td>
<td>30</td>
<td>138</td>
<td>18.07</td>
<td>28 63 25 46 38</td>
<td>290 290 324 250 226</td>
<td>3 34 12 10 10</td>
</tr>
<tr>
<td>IU35</td>
<td>35</td>
<td>183</td>
<td>22.49</td>
<td>13 11 25 17 19</td>
<td>257 202 183 241 222</td>
<td>4 6 19 15 15</td>
</tr>
<tr>
<td>IU40</td>
<td>40</td>
<td>159</td>
<td>25.85</td>
<td>13 14 38 13 18</td>
<td>353 232 292 214 170</td>
<td>5 5 15 10 12</td>
</tr>
<tr>
<td>IU45</td>
<td>45</td>
<td>198</td>
<td>29.82</td>
<td>165 186 158 157</td>
<td>MOut 483 566 564 439</td>
<td>10 39 24 20 22</td>
</tr>
<tr>
<td>IU50</td>
<td>50</td>
<td>615</td>
<td>31.57</td>
<td>476 540 701 427 561</td>
<td>1627 1602 1655 2020 2384</td>
<td>16 77 238 208 250</td>
</tr>
<tr>
<td>IU55</td>
<td>55</td>
<td>625</td>
<td>33.94</td>
<td>982 870 1011 614 585</td>
<td>4683 3298 4923 4189 3100</td>
<td>14 84 322 224 223</td>
</tr>
<tr>
<td>IU60</td>
<td>65</td>
<td>632</td>
<td>39.32</td>
<td>1083 1161 809 751</td>
<td>MOut 6793 6965 6711 5440</td>
<td>18 100 406 414 361</td>
</tr>
<tr>
<td>IU70</td>
<td>70</td>
<td>635</td>
<td>42.07</td>
<td>5398 2855 3596 2371 2947</td>
<td>17355 9964 8225 9619 8570</td>
<td>38 129 943 885 966</td>
</tr>
<tr>
<td>IU75</td>
<td>75</td>
<td>322</td>
<td>46.59</td>
<td>5367 3822 4911 2828 2522</td>
<td>16538 9404 1309 8707 6414</td>
<td>45 140 1395 1118 1057</td>
</tr>
<tr>
<td>IU80</td>
<td>80</td>
<td>350</td>
<td>49.80</td>
<td>6984 5418 4552 4302</td>
<td>MOut 17993 20193 16018 12062</td>
<td>49 136 1975 1728 1964</td>
</tr>
<tr>
<td>IU85</td>
<td>85</td>
<td>362</td>
<td>52.14</td>
<td>6933 5558 6289</td>
<td>MOut 25661 23659</td>
<td>59 154 2633 1950 2764</td>
</tr>
<tr>
<td>TCAS</td>
<td>139</td>
<td>0</td>
<td>106.87</td>
<td>5058 4598 4139 3781 3646</td>
<td>11931 9140 8463 7792 6494</td>
<td>27 165 69 57 70</td>
</tr>
<tr>
<td>S1269</td>
<td>37</td>
<td>18</td>
<td>30.07</td>
<td>2109 1875 1939 1703 1540</td>
<td>1440 893 858 645 538</td>
<td>10 24 331 299 318</td>
</tr>
<tr>
<td>S1512</td>
<td>57</td>
<td>29</td>
<td>40.59</td>
<td>799 651 94 505 431</td>
<td>159 135 167 122 80</td>
<td>15 30 193 140 177</td>
</tr>
<tr>
<td>S3578</td>
<td>179</td>
<td>35</td>
<td>57.71*</td>
<td>18036 10168 10184 8862 8092</td>
<td>1632 1279 1039 936 889</td>
<td>42 67 4539 2958 3851</td>
</tr>
<tr>
<td>S4863</td>
<td>104</td>
<td>49</td>
<td>72.35</td>
<td>3565 3013 3630 2111 1747</td>
<td>1124 910 924 986 663</td>
<td>38 56 653 415 565</td>
</tr>
<tr>
<td>S3271</td>
<td>116</td>
<td>26</td>
<td>79.83</td>
<td>4234 3299 4723 2947 2838</td>
<td>8635 6203 9023 5601 4105</td>
<td>33 30 815 732 710</td>
</tr>
<tr>
<td>S3330</td>
<td>132</td>
<td>40</td>
<td>86.64</td>
<td>23659 24563</td>
<td>MOut 18893 16946</td>
<td>12837 11381 MOut 9927 7626</td>
</tr>
<tr>
<td>S3271</td>
<td>116</td>
<td>26</td>
<td>79.83</td>
<td>4234 3299 4723 2947 2838</td>
<td>8635 6203 9023 5601 4105</td>
<td>33 30 815 732 710</td>
</tr>
<tr>
<td>S3330</td>
<td>132</td>
<td>40</td>
<td>86.64</td>
<td>23659 24563</td>
<td>MOut 18893 16946</td>
<td>12837 11381 MOut 9927 7626</td>
</tr>
</tbody>
</table>

Table 1: Comparing our three static algorithms VS-I, VS-II and VS-III against FMCAD00 and Simulated annealing (SA) algorithm. (MOut) – Out of memory, (*)–SFEISTEL, (**)–after 8 reachability steps, (***)–after 14 reachability steps. Total time includes the time for the static phase. The circuits could run out of memory at different stages (marked as “memory out”), so results are provided for the stages that were finished, e.g, for s1423, VS-III run out memory during reachability analysis, so the BDD Nodes statistic is for initial phase.


