# Implicative Simultaneous Satisfiability and Applications

Zurab Khasidashvili and Alexander Nadel

Intel Corporation, P.O. Box 1659, Haifa 31015 Israel
{zurab.khasidashvili,alexander.nadel}@intel.com

**Abstract.** This paper proposes an efficient algorithm for the systematic learning of implications. This is done as part of a new search and restart strategy in the SAT solver. We evaluate the new algorithm within a number of applications, including BMC and induction with invariant strengthening for equivalence checking. We provide extensive experimental evidence attesting to a speedup of one and often two orders of magnitude with our algorithm, on a representative set of industrial and publicly available test suites, as compared to a basic version of invariant strengthening. Moreover, we show that the new invariant strengthening algorithm alone performs better than induction and interpolation, and that the absolutely best result is achieved when it is combined with interpolation. In addition, we experimentally demonstrate the superiority of an application of our new algorithm to BMC.

## 1 Introduction

The need to efficiently solve many closely related problems arises in numerous applications of model checking [8] and equivalence checking [12]. Various automatic invariant strengthening algorithms fall into this class of applications. In such algorithms one has to guess the missing invariants that strengthen the target property, thereby making it easier to prove. However, for the guessing to succeed, many potential invariants must be tried out, and therefore for overall efficiency it is very important that the evaluation of potential invariants be very fast.

In the domain of SAT solving [3, 14], several efficient approaches to solving multiple related objectives *incrementally* [26, 27] or *simultaneously* [13] have been developed in the past, and due to its increasing importance this is an active research area. The two approaches are closely related, yet there are subtle fundamental differences between the two. Their relative performance depends on the nature of the benchmarks.

In this work we focus on improving and refining the *Simultaneous SATisfiability* (or *SSAT*) approach to solving multiple closely related SAT tasks. Recall that the *SSAT* algorithm aims at proving a number of related objectives (called *proof objectives*, or POs) in *one* backtrack search. The algorithm receives a CNF instance and a number of literals that occur in the CNF. These literals represent the POs. If there is a satisfying assignment to the CNF where a PO is assigned false, then the PO is not a logical consequence of the CNF instance and is therefore called *falsifiable*; otherwise it is *valid*. For *each* PO, the $SSAT$ algorithm returns one of the following statuses: *valid*, *falsifiable*, or *indeterminate* (in case the algorithm is interrupted). This is different from solving the satisfiability status of the conjunction of all POs.

As an example, consider the problem of combinational or sequential equivalence checking of circuits. By the nature of the problem, and in particular, by the nature of the design or synthesis of an implementation model based on a specification model, there are many internal nodes in the circuits that are equivalent. Exploiting these internal equivalences often helps enormously in proving the functional equivalence of the corresponding outputs of the two circuit designs. The POs are then equivalences of the form $x \leftrightarrow y$ or $x \leftrightarrow \neg y$ between internal nodes $x$ and $y$ of the specification and implementation circuits. On a large set of Intel and academic benchmarks, we found that $SSAT$ is more efficient within the invariant-strengthening algorithms than saturation or multiple incremental calls to the SAT solver. Still, there is an inefficiency caused by the fact that the definitions of many (sometimes tens or hundreds of thousands) of the POs corresponding to the candidate invariants must be added to the CNF instance: an equivalence of the form $x \leftrightarrow y$ or $x \leftrightarrow \neg y$ is translated into four clauses (with the standard Tseitin encoding), and these extra clauses noticeably slow down the SAT solver.

We introduce a novel DPLL-based approach, called implicative $SSAT$ (or $SSAT^{\rightarrow}$), that leverages from the fact that the POs are equivalences consisting of two implications, e.g., $x \rightarrow y$ and $y \rightarrow x$. The $SSAT^{\rightarrow}$ algorithm learns these implications and equivalences *without encoding them into the CNF*. Instead, it deals with them during the search using a dedicated algorithm. This leads to a speedup of up to two orders of magnitude as compared to other approaches to invariant strengthening that also try to prove a maximal number of candidate invariants at a given induction depth. Our algorithm can solve any number of user-given properties simultaneously. The algorithm is discussed in Section 2.

We propose two new applications of $SSAT$ and $SSAT^{\rightarrow}$ (Sections 3 and 4, respectively). One is *in-depth BMC*, which uses simultaneous solving in a BMC [2] scheme where unrolling happens with intervals [11, 28] (the SAT solver is not called after each unrolling step). The other application is an invariant strengthening algorithm for equivalence checking known as van Eijk's method [10]. Both Sections 3 and 4 present a rich collection of experiential results demonstrating the efficiency of our algorithms. In Section 5 we provide an extensive overview of related work, in order to make it clear how our research advances to the state of the art. Conclusions and discussion of future work appear in Section 6.

## 2 Implicative $SSAT$

Recall that $SSAT$ modifies the modern SAT solver's algorithm in a way that allows it to solve multiple proof objectives in one search ([13], Section 5). The $SSAT$ algorithm always maintains a PO literal, called the currently watched PO (CWPO), that the SAT search tries to falsify. At the beginning of the search CWPO is set to be any PO literal. At every stage of the search, prior to invoking a generic decision heuristic, CWPO is assigned false. The CWPO ceases to be the currently watched PO in two circumstances:

(1) When a model containing CWPO = false is discovered, in which case we mark as falsifiable the CWPO as well as all the POs that are assigned false (or are don't care literals) in the model;
(2) When the CWPO is discovered to be globally true, in which case we mark the CWPO as valid.

In either of these circumstances, we check whether there exists an *unresolved* PO $l$ – a PO that has not been found valid or falsifiable. If such an unresolved PO $l$ exists, we set the CWPO to $l$, otherwise the algorithm halts. The algorithm returns the pair $(vPOs, fPOs)$ consisting of POs proved valid and POs proved falsifiable.

We found it very useful to frequently reschedule POs in the $SSAT$ algorithm: each CWPO ceases to be a CWPO after a given number of restarts, and the next PO in a sorted list of POs is selected as the CWPO. In other words, the list of unresolved POs is rotated. This is different from the original $SSAT$ algorithm (where a CWPO ceases to be a CWPO only after it gets resolved), and often prevents wasting search effort in irrelevant search space: the learning gained resolving other POs often makes it easier to resolve the once problematic CWPO later. In particular, thanks to frequent rescheduling, simpler invariants are discovered easily and solved first; rescheduling can thus be seen as an improved version of the widely used method according to which candidate invariants are sorted in a bottom-up fashion and solved in that order.

As discussed earlier, in applications where the POs are equivalences of the form $PO = o_s \leftrightarrow o_i$, and there are many POs, translating them all into the CNF instance can be a significant overhead for the solver. Therefore, we propose implicative $SSAT$, or $SSAT^{\rightarrow}$, as an algorithm that takes a number of pairs $(o_s^j, o_i^j)$ as input and reports the status of each equivalence $o_s^j \leftrightarrow o_i^j$ for each $j$: if there is a satisfying assignment with $\neg o_s^j \wedge o_i^j$ or $o_s^j \wedge \neg o_i^j$, then the equivalence is false and $PO^j = o_s^j \leftrightarrow o_i^j$ is *falsifiable*; otherwise it is *valid*.

For deciding the validity of an equivalence $PO = o_s \leftrightarrow o_i$, the algorithm checks the satisfiability status of two implications $PO^{\rightarrow} : o_s \rightarrow o_i$ and $PO^{\leftarrow} : o_s \leftarrow o_i$. A pseudo algorithm for $SSAT^{\rightarrow}$ is described in Figure 1. For simplicity of presentation, we do not treat the circumstance where there are initial unit clauses, in which case some POs might be found valid before the loop at line 2. The algorithm's structure is similar to that of SSAT. The main difference is that the $SSAT^{\rightarrow}$ algorithm needs to track the status of implications, rather than single literals. Consequently, the treatment of the CWPO in lines 8 – 19 becomes more complex, since each PO has two implications and each implication has two literals. Consider line 25, which is supposed to find globally valid PO implications. Our algorithm (not specified in Figure 1) returns that a PO implication $PO^{\rightarrow} : o_s \rightarrow o_i$ is valid if one of the following conditions holds:

(1) $o_s$ is globally false (false at decision level 0);
(2) $o_i$ is globally true;
(3) At decision level 1: both $o_s$ and $o_i$ are true, where $o_s$ is the decision literal and $o_i$ is an implied literal.

## 3 In-depth BMC with $SSAT^{\rightarrow}$

In this section we discuss a variant of the BMC algorithm that employs $SSAT^{\rightarrow}$ in a way that differs from the known usages of incremental SAT with assumptions in BMC [9]. Besides the maximal bound $k$, *BMC with intervals* [11] takes an argument $i$ that denotes the length of the bound intervals in which SAT checks for falsification of the property are performed. For instance, with $i = 10$ and $k = 100$, bound intervals $0-9, 10-19, \ldots, 90-99, 100$ are checked consecutively and incrementally. More precisely, given a safety property $P$ and a state $s$, assume that $P(s)$ is a variable denoting $P$ in state $s$. Then in the interval $0-9$, BMC with intervals calls the SAT solver to check the satisfiability of the following

```
$SSAT^{\rightarrow}$ ($cnf$, [$PO_1 = (o_s^1, o_i^1), \ldots, PO_n = (o_s^n, o_i^n)$])
 1: CWPO = any PO pair;
 2: while (true) do
 3:   if CWPO is marked valid or falsifiable then
 4:     if all the POs are marked valid or falsifiable then
 5:       return ($vPOs, fPOs$);
 6:     end if
 7:     CWPO = any PO pair $(o_s^j, o_i^j)$ that is yet unresolved;
 8:     if $PO^{\rightarrow}$ is unresolved then
 9:       $\sigma$ = true
10:     else
11:       $\sigma$ = false
12:     end if
13:     if $o_s$ is unassigned then
14:       Assign $o_s = \sigma$
15:     else
16:       if $o_i$ is unassigned then
17:         Assign $o_i = \neg\sigma$
18:       end if
19:     end if
20:   else
21:     Assign choose-decision-literal();
22:   end if
23:   while (status == local-conflict) do
24:     status = BCP();
25:     Mark any PO implication $PO^{\rightarrow}$ or $PO^{\leftarrow}$ that is discovered to be globally
        true as valid;
26:     If for an unresolved PO both $PO^{\rightarrow}$ and $PO^{\leftarrow}$ are marked valid, mark the
        PO valid;
27:     if status == global-conflict then
28:       Mark all unmarked POs valid;
29:       return ($vPOs, fPOs$);
30:     end if
31:     if (status == model) then
32:       Mark any falsified PO implication $PO^{\rightarrow}$ or $PO^{\leftarrow}$ falsifiable;
33:       If for an unresolved PO one of the implications $PO^{\rightarrow}$ and $PO^{\leftarrow}$ is marked
          falsifiable, mark the PO falsifiable;
34:       Unassign all the literals that are not globally true;
35:     end if
36:     if (status == local-conflict) then
37:       Add a conflict clause; Backtrack;
38:       Assign literal that must be flipped following conflict analysis;
39:     end if
40:   end while
41: end while
```

**Fig. 1.** Pseudo algorithm for implicative $SSAT$ (or $SSAT^{\rightarrow}$).

| Family | BMC | BMC10 | BMC10$^{\rightarrow}$ | BMC25 | BMC25$^{\rightarrow}$ | ABC-BMC2 | ABC-BMC3 |
|---|---|---|---|---|---|---|---|
| bj | 405 | 393 | 461 | 328 | 462 | 485 | 553 |
| bob | 674 | 645 | 736 | 427 | 697 | 706 | 710 |
| cmu | 137 | 140 | 138 | 150 | 143 | 158 | 174 |
| eij | 785 | 765 | 846 | 551 | 848 | 585 | 730 |
| nus | 283 | 301 | 301 | 301 | 355 | 225 | 385 |
| pdt | 3393 | 3504 | 3963 | 2957 | 3983 | 3430 | 3673 |
| pj | 300 | 312 | 331 | 252 | 347 | 404 | 404 |
| texas | 202 | 202 | 168 | 176 | 199 | 202 | 202 |
| Total (bound) | 6179 | 6262 | 6944 | 5142 | **7034** | 6195 | 6831 |

**Table 1.** Comparing the bound for 115 timed-out instances.

| Family | BMC | BMC10 | BMC10$^{\rightarrow}$ | BMC25 | BMC25$^{\rightarrow}$ | ABC-BMC2 | ABC-BMC3 |
|---|---|---|---|---|---|---|---|
| bj | 215.8 | 324.9 | 190.6 | 341.4 | 103.3 | 407.07 | 51.08 |
| bob | 116.7 | 853.1 | 648.9 | 1622 | 1176 | 186.99 | 174.61 |
| cmu | 203.4 | 21.1 | 14.6 | 21.9 | 17.3 | 3.9 | 3.38 |
| eij | 678.2 | 513.1 | 190.2 | 1446.1 | 141.4 | 62.38 | 171.58 |
| nus | 2609.1 | 747.5 | 617 | 933.4 | 883.8 | 311.87 | 396.16 |
| pdt | 4202 | 2837.1 | 1981.8 | 4230.3 | 1884.8 | 1350.72 | 955.41 |
| pj | 513.2 | 911.1 | 449 | 829.4 | 555.8 | 569.72 | 940.1 |
| texas | 8.4 | 22.4 | 142.1 | 33.7 | 37.7 | 25.67 | 25.06 |
| vis | 311.9 | 238 | 138.7 | 406.3 | 147.6 | 94.7 | 169.52 |
| Total (cpu time) | 9614.6 | 8049.7 | 5187.5 | 12336.5 | 5697.9 | 4458.19 | **3108.8** |

**Table 2.** Comparing the run-time for 302 completed instances.

formulas, where $Tr$ and $I$ denote the transition and initial state relations, and $P_{0-9} = P(s_0) \wedge \ldots \wedge P(s_9)$.

$$I(s_0) \wedge path(s_0, \ldots, s_9) \wedge \neg P_{0-9}$$
$$path(s_0, \ldots, s_k) = Tr(s_0, s_1) \wedge \ldots \wedge Tr(s_{k-1}, s_k)$$

In the *in-depth BMC* algorithm that we propose, in each interval such as $0-9$, we call $SSAT^{\rightarrow}$ with POs $P(s_0), \ldots, P(s_9)$, on the unrolled instance.

Tables 1 and 2 compare our implementations of incremental BMC [9] (column BMC), incremental BMC with intervals (BMC10, BMC25), and in-depth BMC (BMC10$^{\rightarrow}$, BMC25$^{\rightarrow}$), for the maximal bound $k = 100$ and intervals $10, 25$, on 417 problems from several families of HWMCC'10 benchmarks. We also compare our results with ABC-BMC3, which is ABC's implementation of incremental BMC, and ABC-BMC2, which is similar to BMC with intervals but whose unrolling intervals are determined based on the extra gate count [19]. We selected the problems that were unsatisfiable for both the ABC-BMC2 strategy in the competition and our BMC implementation, since most of the falsifiable instances in the HWMCC'10 set are too easy. The first table shows 115 problem instances for which a 900-second time-out occurred for at least one of the strategies before the maximal bound was reached; for each group the sum of the reached bounds is shown. The second table shows the run-times per group and per strategy for the remaining 302 problem instances.

The tables show that in-depth BMC reaches higher bounds than any other version of BMC. Implication learning was disabled in these experiments, since for BMC these techniques are useful only for very difficult instances. The advantage of in-depth BMC over BMC can be explained as follows: In the interval $0-9$, while trying to falsify, say, $P(s_3)$, the $SSAT^{\rightarrow}$ solver has a view of the cones of $P(s_4),\ldots,P(s_9)$ as well; this allows the solver to infer and use useful correlations among signals at bounds up-to 3 from the definitions, user constraints (possibly sequential), and PO assignments at higher bounds. While BMC with intervals also has the view of the cones of all the POs $P(s_o),\ldots,P(s_9)$, in contrast to in-depth BMC, it can either solve them all together (by proving that $\neg P_{0-9}$ is unsatisfiable) or solve none of them. Solving all the POs is more complex than only proving valid POs up to $P(s_3)$.

## 4 $SSAT^{\rightarrow}$ and strengthening inductive invariants

A basic scheme combining invariant strengthening and temporal induction is depicted in Figure 2. This scheme was proposed in [4]. The induction and invariant strengthening algorithms in [23, 10], as well as the algorithms that we propose, can be seen as instances of this algorithm scheme. Recall that, according to the temporal induction scheme [23], a property $P$ is valid if for some $m$, the formulas $base(P,k)$ and $step(P,m)$ defined below are unsatisfiable for all $0 \leq k \leq m$ (for simplicity, we omit discussion of the loop-free condition for the induction step).

$$base(P,k) = I(s_0) \wedge path(s_0,\ldots,s_k) \wedge P(s_0) \wedge \ldots \wedge P(s_{k-1}) \wedge \neg P(s_k)$$
$$step(P,k) = path(s_0,\ldots,s_{k+1}) \wedge P(s_0) \wedge \ldots \wedge P(s_k) \wedge \neg P(s_{k+1})$$

To interface the base and step formulas $base(P,k)$ and $step(P,k)$ for a property $P = l \leftrightarrow r$ with $SSAT^{\rightarrow}$, we change them as follows, where $l(s_i) \leftrightarrow r(s_i)$ represents $P$ at state $s_i$:

$$base\_cnf(k) = I(s_0) \wedge path(s_0,\ldots,s_k)$$
$$base\_PO\_pairs(P,k) = [(l(s_k), r(s_k))]$$
$$step\_assumption(P,k) = P(s_0) \wedge \ldots \wedge P(s_k)$$
$$step\_cnf(k) = path(s_0,\ldots,s_{k+1})$$
$$step\_assum\_cnf(P,k) = step\_cnf(k) \wedge step\_assumption(P,k)$$
$$step\_PO\_pairs(P,k) = [(l(s_{k+1}), r(s_{k+1}))]$$

Then, $base(P,k)$ is satisfiable iff

```
**Induction with invariant strengthening** $(P, nMaxDepth)$
  k = 0;
  $POs = Create\_candidate\_invariants()$ $(where\ P \in POs)$;
  **while** $(k <= nMaxDepth)$ **do**
    $POs = BASE(POs, k)$;
    **if** $(POs \neq [\ ])$ **then**
      $POs = STEP(POs, k)$;
    **end if**
    **if** $(POs \neq [\ ])$ **then**
      $k{+}{+}$;
    **end if**
  **end while**
```

**Fig. 2.** Pseudo algorithm for induction with invariant strengthening.

$$SSAT^{\rightarrow}(base\_cnf(k), base\_PO\_pairs(P,k))$$

returns $P$ as falsifiable; similarly *step*$(P,k)$ is unsatisfiable iff

$$SSAT^{\rightarrow}(step\_assum\_cnf(P,k), step\_PO\_pairs(P,k))$$

returns $P$ as valid. Thus the above formulas define a sound way of using $SSAT^{\rightarrow}$ in temporal induction.

Now, if we want to perform the base and step checks simultaneously for a number of POs $PO_0, \ldots, PO_n$, the definitions of $base\_cnf(k)$ and $step\_cnf(k)$ remain unchanged – they do not depend on the properties that one is interested in. The step assumption is the conjunction of formulas $step\_assumption(PO_i, k)$ for $0 \leq i \leq n$, and the base and step PO lists are defined as follows:

$$\begin{array}{l}
base\_PO\_pairs(POs,k) = [(l_0(s_k), r_0(s_k)), \ldots, (l_n(s_k), r_n(s_k))] \\
step\_PO\_pairs(POs,k) = [(l_0(s_{k+1}), r_0(s_{k+1})), \ldots, (l_n(s_{k+1}), r_n(s_{k+1}))] \\
step\_assumption(POs,k) = \wedge_{i=0}^{i=n} step\_assumption(PO_i, k) \\
step\_assum\_cnf(POs,k) = step\_cnf(k) \wedge step\_assumption(POs,k)
\end{array}$$

The $BASE(POs,k)$ procedure employing $SSAT^{\rightarrow}$, called *base_issat*, is depicted in Figure 3.(a). $vPOs$ denote the POs whose corresponding base formulas are valid in state $s_k$. Similarly, Figure 3.(b) depicts the $STEP(POs,k)$ procedure employing $SSAT^{\rightarrow}$, called *step_issat*.

We call the invariant strengthening algorithm just described $invSSAT^{\rightarrow}$. In the same vein, we refer to the similar invariant strengthening algorithm employing $SSAT$ as $invSSAT$. Further, in the experiments reported below, we use $invSATURk$ to denote the invariant strengthening scheme

```
(a) base_issat ($POs, k$)
  $(vPOs, fPOs) = SSAT^{\rightarrow}(base\_cnf(k), base\_PO\_pairs(POs, k))$;
  report $fPOs$ as falsifiable;
  return vPos;
(b) step_issat ($POs, k$)
  $i = 0$;
  $POs^0 = POs$;
  while (true) do
    $(vPOs^i, fPOs^i) = SSAT^{\rightarrow}(step\_assum\_cnf(POs^i, k), step\_PO\_pairs(POs^i, k))$;
    $POs^{i+1} = vPOs^i$;
    if ($fPOs^i = [\,]$) then
      break;
    end if
    $i = i + 1$;
  end while
  report $POs^i$ as valid;
  return $POs \setminus POs^i$;
```

**Fig. 3.** $BASE(POs, k)$ and $STEP(POs, k)$ for invariant strengthening with $SSAT^{\rightarrow}$.

of [4], where $k$-saturation is used as combinational reasoning engine. By $invCONJ$, we refer to the invariant strengthening scheme where saturation is replaced by SAT. That is, in $invCONJ$, the candidate invariants are proven using the conjunction approach to verifying multiple properties presented in [11].

| | **Fam. 1** | | **Fam. 2** | | **Fam. 3** | | **Fam. 4** | | **Fam 5** | | **Fam. 6** | | **Fam. 7** | | **Total** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Algorithm** | **time** | **solv** | **time** | **solv** | **time** | **solv** | **time** | **solv** | **time** | **solv** | **time** | **solv** | **time** | **solv** | **time** | **solv** |
| invSSAT$^{\rightarrow}$ & interp. | 8 | 132 | 366 | 574 | 39 | 659 | 1 | 142 | 475 | 147 | 513 | 3775 | 2983 | 230 | **4387** | **5659** |
| invSSAT$^{\rightarrow}$ | 9 | 132 | 365 | 574 | 36 | 659 | 1 | 142 | 16 | 147 | 512 | 3775 | 3686 | 230 | 4627 | 5659 |
| invSSAT | 6 | 132 | 418 | 574 | 45 | 659 | 1 | 142 | 14 | 147 | 258 | 3775 | 21311 | 218 | 22055 | 5647 |
| invSATUR0 & interp. | 94 | 132 | 4345 | 529 | 115 | 659 | 2 | 142 | 221 | 147 | 1841 | 3774 | 683 | 230 | 7304 | 5613 |
| invSATUR0 & induction | 5 | 132 | 10390 | 532 | 675 | 654 | 1000 | 140 | 1000 | 145 | 55 | 3773 | 26059 | 217 | 39186 | 5593 |
| invCONJ | 97 | 132 | 1233 | 574 | 3783 | 659 | 10 | 142 | 61 | 147 | 2077 | 3775 | 229856 | 1 | 237120 | 5430 |
| interp. | 4018 | 128 | 21171 | 494 | 20487 | 548 | 38 | 142 | 3943 | 45 | 4526 | 3774 | 19349 | 217 | 73535 | 5348 |
| induction | 18020 | 43 | 24836 | 478 | 52737 | 185 | 1000 | 133 | 2037 | 131 | 70159 | 184 | 60053 | 187 | 228845 | 1341 |
| invSATUR1 | 11 | 40 | 283 | 19 | 44 | 408 | 2 | 13 | 12 | 18 | 305 | 53 | 752 | 28 | 1413 | 579 |

**Table 3.** Comparing invariant strengthening algorithms as well as strategy combinations with induction and interpolation.

In Table 3 we report experimental results for 7 families of equivalence checking problems from recent microprocessor designs at Intel. Each family corresponds to a functional module. These 7 families together contain

5659 sequential equivalence checking problem instances. The designs are non-state-matching, therefore provable internal equivalences form a very small portion of all potential equivalences (the number of provable equivalences is typically a few hundreds, and many of them are very simple.) The timeout used in our experiments was 1000 seconds (in combined algorithms, the timeout was divided between the strategies). The maximal unrolling bound in all algorithms was 100. In the columns named *solv* we report the number of problem instances solved by each algorithm per family and in total. In the columns named *time* we report the time spent on each family and in total. We compare the different versions of invariant strengthening algorithms discussed in this section, as well as combinations of the best invariant strengthening algorithms with temporal induction [23] and interpolation [18]. We used a rescheduling rate of 5 in the experiments for $invSSAT^{\rightarrow}$.

$invSATUR0$ and $invSATUR2$ could not solve any of the problem instances, and therefore these algorithms do not appear in the table. Compared to the SAT-based conjunction method $invCONJ$, employing $SSAT$ in $invSSAT$ yields an order of magnitude speedup, and an additional 217 problem instances were solved. Employing $SSAT^{\rightarrow}$ in $invSSAT^{\rightarrow}$ yields an average $52.6x$ speedup compared to $invCONJ$, and an additional 229 problems were solved. The additional gain in $invSSAT^{\rightarrow}$ compared to $invSSAT$ is due to quick processing of candidate implications in $SSAT^{\rightarrow}$, which was our main motivation for introducing $SSAT^{\rightarrow}$. While $invSATUR0$ is too weak to solve any of the problem instances, it can very quickly prove important equivalences which might help the induction [23] and interpolation [18] algorithms. Similarly, invariants proven by $invSSAT^{\rightarrow}$ can significantly improve the runtime of interpolation. The results show that $invSSAT^{\rightarrow}$ outperforms both the induction and interpolation algorithms, and that $invSSAT^{\rightarrow}$ combined with interpolation is the winning algorithm overall: it solves all 5659 problem instances, and is faster than any other single or combined algorithm.

We also ran several strategy combinations on the 417 problems discussed in Section 3, and compared them with their counterparts in ABC (ABC-scorr denotes ABC's scorr algorithm with unrolling bound 100; scorr is the algorithm within ABC that is the closest to our invariant strengthening algorithms). The summary of these results is presented in Table 4: as one can see, the results, both in terms of solved problems and runtimes, are similar to what was observed on Intel benchmark families, although the impact of combining invariant strengthening strategies with induction and interpolation algorithms is even greater. The combination

| Algorithm | time | solv |
|---|---|---|
| invSSAT$^{\rightarrow}$& interp. | **78106** | **319** |
| ABC-interpolation | 109166 | 310 |
| invSSAT$^{\rightarrow}$ | 133266 | 296 |
| invCONJ | 135519 | 294 |
| invSSAT | 135774 | 294 |
| invSATUR0& interp. | 141619 | 283 |
| interpolation | 144131 | 282 |
| ABC-scorr & ABC-interp | 149540 | 281 |
| invSATUR0& induction | 147645 | 265 |
| ABC-scorr | 146685 | 242 |
| induction | 176056 | 231 |

**Table 4.** Comparing strategy combinations on 417 HWMCC'10 problem instances.

of $invSSAT^{\rightarrow}$ followed by interpolation (which reuses the learning) remains the best combination.

## 5 Related work

The CNF and the POs for an $SSAT^{\rightarrow}$ instance are often produced by translating a circuit *ckt* into CNF. Following [22], in our model-checking tool, a circuit is represented as a collection of *triplets* of the form $x := y \rightarrow z$ or $x := y \leftrightarrow z$, where $x$, $y$, and $z$ are literals. This is close to the widely used And-Inverter Graph (AIG) representation [16], where the triplets are of the form $x := y \wedge z$. Besides the structural information recorded as triplets, our representation of *ckt* maintains its variables in equivalence classes ($E$-classes, for short) [22]; each class has a representative variable, and all the variables in an equivalence class are known to be (logically) equivalent to the representative or its negation. When converting *ckt* into an $SSAT$ instance, only the representative variables and the relations between them are reflected in the resulting CNF. The unit and two-literal clauses learned during the $SSAT$ search are added to *ckt*. Saturation [24] on the enhanced *ckt* may yield additional learning, including the learning of equivalences and inverse equivalences among the representative variables of $E$-classes; this in turn enables merging $E$-classes and reducing the number of representative variables.

The aim of $SSAT^{\rightarrow}$ is to solve *closely related* objectives in at most one complete search. The $SSAT^{\rightarrow}$ solver has a view of the *entire problem instance*, and has the freedom to focus on resolving a particular objective and to switch between the objectives *as part of the search strategy*. Furthermore, again as part of the search strategy, $SSAT^{\rightarrow}$ tries to learn the entire instance by learning the implications between all pairs of variable assignments. These implications are recorded as two-literal clauses (which are known to be very important learnings). Equivalences derived from them enable merging variables by merging their equivalence clauses.

The idea of learning the implications between circuit signal values was introduced in the *recursive learning* algorithm [17], in the context of circuit ATPG [1]. There implications are learned iteratively using a dedicated constant propagation algorithm, with increasing effort at each iteration. The learned implications are used to speedup the backward justification process (which is the main routine of ATPG). $SSAT^{\rightarrow}$ can be seen as a SAT-based implementation of recursive learning, where the learning of implications happens as part of the SAT search rather than as a separate routine. Indeed, $SSAT^{\rightarrow}$ introduces a new dimension to restart strategies. Traditionally, a restart strategy refers to *when* to restart, not *how* to restart. $SSAT$ introduces *fairness* into the how-to-restart strategy: the first two variable assignments after a restart do not follow the default search heuristic of the SAT solver; instead, every pair of variables in the candidate equivalences list is considered with four possible value assignments. We note that the *when* part of the restart strategy in our $SSAT^{\rightarrow}$ solver is the same in all the reported experiments; and our SAT solver is implemented as a special case of $SSAT^{\rightarrow}$ since the latter has more generic (incremental) interface (API).

The crucial idea of simplifying equivalence checking by proving (observable) internal equivalences and merging equivalent nodes was introduced in [6]. The AIGs data structure and BDD and SAT sweeping [16, 15] allow for very efficient implementation of this idea. The triplet and equivalence classes data structure is closely related to the AIGs data structure, however it is no longer a DAG. In addition we work with constraints explicitly, a fact which entangles the cones of the objectives even more tightly. Finally, we do not use local BDDs [16] or AIG rewriting [5] to optimize the problem instance; instead we rely on saturation [24] and on learning from $SSAT^{\rightarrow}$ to achieve a compact representation.

An incremental version of SAT sweeping was proposed in [19], where, in addition, rescheduling of candidate equivalences was first considered. In that paper the authors work incrementally with a *SAT-with-assumptions* interface [9]. The idea behind $SSAT^{\rightarrow}$ could be used to extend SAT-with-assumptions, so that it could treat assumptions that are implications rather than literals.

Unlike $SSAT^{\rightarrow}$, in [19] each CWPO is targeted for falsification in its *cone of influence*; this is achieved by an API that allows the SAT solver to work with a subset of relevant variables (computed based upon the circuit), and Boolean Constraint Propagation (BCP) needs to be modified accordingly. This modified search procedure is not described in detail in [19], we therefore couldn't re-implement it for a fair comparison (moreover, recall that we use a very different circuit representation). Our early experience with simultaneous solving of multiple POs by solving each PO in its cone of influence and re-using *pervasive* learned clauses compared to $SSAT$ is reported in [13], and is negative. In fact, one of the main original motivations for introducing $SSAT$ was to eliminate the overhead of computing the cones of each objective and managing the conflict clauses. The relative performance of these two methods certainly depends on the nature of the problems at hand; Overall, our experience (within our implementation) is that $SSAT$ performs much better when the POs that are solved simultaneously are closely related (and their cones have a high percentage of overlap). As observed in [15], *modern SAT solvers are efficient in focusing on relevant parts of the problem.* This saves us the effort of *forcing* the SAT solver to work exclusively with the cone-of-influence of the CWPO. Furthermore, when working with the entire instance, one has a greater freedom in deciding assignments for other PO implications (that might not be in the cone), thereby increasing the chance of solving them as a side-effect of the search. For example, in $SSAT^{\rightarrow}$, the default behavior is that after assigning the CWPO, the other yet unresolved user POs are assigned next, with false. Experimental evaluation of the in-depth BMC strategy clearly demonstrates that giving a solver a wider view of the problem instance and letting it decide how to perform the search is beneficial.

Since in $SSAT^{\rightarrow}$, unlike in previous work, solving candidate equivalences is a by-product of the search heuristic, in our approach it becomes much less important to reduce candidate equivalences by quick falsification methods such as simulation with random or biased input patterns or satisfying-assignments [6, 15, 19]. Since we work with user constraints explicitly, as part of the E-classes data structure, we cannot use random simulation of inputs for quick falsification (the constraints need not hold for arbitrary input assignments). We have experimented with multiple methods of diverse satisfying assignment generation [21] in order to use them as simulation patterns for quick falsification of candidate equivalences, and while they can significantly reduce the number of candidate equivalences, this didn't noticeably affect overall runtime, because $SSAT^{\rightarrow}$ typically generates many satisfying assignments during the search (biased towards the falsification of as many POs as possible) and they filter out false candidate equivalences very efficiently.

The basic idea of invariant strengthening for sequential equivalence checking was proposed by van Eijk in [10]. The transition invariants were computed using BDDs. This idea was further generalized in [4], in two ways: the basic transition invariant scheme was enhanced by the temporal induction scheme [23], and saturation [24] replaced the usage of BDDs. Numerous circuit-level optimizations were proposed in [7]; the main differences with our approach has been discussed above. The *speculative reduction* technique [20] further advances van Eijk's method by strengthening the inductive assumptions within refinement iterations of candidate invariant set; this is done by creating copies of the current set of candidate invariants and assuming them (*in all reachable states*) when proving other candidates.

## 6 Conclusions

The main contribution of this paper is the introduction of a highly scalable and efficient DPLL-based algorithm $SSAT^{\rightarrow}$ that can decide the satisfiability of a large number of (user-given and automatically generated) proof objectives in a single DPLL search, where each proof objective can be either a single literal or an implication between two literals.

We have presented a number of applications of $SSAT^{\rightarrow}$ in bounded and unbounded model checking. The experimental results on academic as well as Intel benchmarks for in-depth BMC and for induction with invariant strengthening fully support the usefulness of these new algorithms compared to the state-of the art.

The $SSAT^{\rightarrow}$ algorithm has already been used as an efficient core DPLL-based engine in many other verification applications at Intel.

Our implication learning algorithms can be viewed as advanced techniques for simplifying combinational problems by systematically learning 2-literal clauses. An interesting future work would be to investigate how to deal with more complex relations between pairs or triplets of literals using a dedicated DPLL-based algorithm. In particular, an immediate generalization of the idea of $SSAT^{\rightarrow}$ would be to designate an algorithm that would efficiently learn 3-literal clauses as part of the SAT search: for example, if no implications have been learned between variables $a, b, c$, a clause $a \wedge \neg b \wedge c$ can be learned if any other combination of the assignments leads to a global conflict.

***Acknowledgements*** We thank the ABC developers for help in running the ABC experiments reported in the paper. Baruch Sterin contributed to the ideas and the implementation of some of the reported algorithms.

## References

1. Abramovici A., M.A., Breuer, A.D. friedman. *Digital Systems Testing and Testable Design*, Computer Science Press, 1990.
2. Biere A., A. Cimatti, E. Clarke, Y. Zhu. Symbolic model checking without BDDs, TACAS 1999.
3. Biere A., M. Heule, H. Van Maaren, T. Walsh. *Handbook of Satisfiability*, IOS Press, 2009.
4. Bjesse P., Claessen C. SAT based verification without state space traversal, FM-CAD 2000.
5. Bjesse P., A. Boralv. DAG-aware circuit compression for formal verification, IC-CAD 2004.
6. Brand D. Verification of large synthesized designs, ICCAD 1993.
7. Case M. L., A. Mishchenko, R. K. Brayton, J. Baumgartner, and H. Mony. Invariant-strengthened elimination of dependent state elements, FMCAD 2008.
8. Clarke E.M., O. Grumberg, D.A. Peled. *Model Checking*, MIT Press, 1999.
9. Eén N., Sörensson N. Temporal induction by incremental SAT solving, ENTCS 89(4), 2003.
10. van Eijk, C.A.J. Sequential equivalence checking without state space traversal, DATE 1998.
11. Fraer, R., S. Ikram, G. Kamhi, T. Leonard, A. Mokkedem. Accelerated verification of RTL assertions based on satisfiability solvers, HLDVT 2002.
12. Huang, S.-Y., K.-T., Cheng. *Formal Equivalence Checking and Design Debugging*, Kluwer, 1998.
13. Khasidashvili, Z., A. Nadel, A. Palti, Z. Hanna. Simultaneous SAT based model checking of safety properties, HVC 2005.
14. Kroening D., Strichman O. *Decision Procedures*, Springer EATCS, 2008.
15. Kuehlmann, A. Dynamic Transition Relation Simplification for Bounded Property Checking, ICCAD 2004.
16. Kuehlmann A., F. Krohm. Equivalence checking using cuts and heaps, DAC 1997.
17. Kunz W., D. K. Pradhan. Recursive learning: An attractive alternative to the decesion tree for test generation in digital circuits, ITC 1992.
18. McMillan, K. L. Interpolation and SAT-based model checking, CAV 2003.
19. Mishchenko A., S. Chatterjee, R. Brayton, N. Een. Improvements to combinational equivalence checking, ICCAD 2006.
20. Mony H., J. Baumgartner, A. Mishchenko, R. Brayton. Speculative reduction-based scalable redundancy identification, DTAE 2009.
21. Nadel A. Generating diverse solutions in SAT, SAT 2011.
22. Nordström J.. Stålmarck's Method Versus Resolution: A Comparative Theoretical Study, Stockholm University, 2001.
23. Sheeran, M. S. Singh, G. Stälmarck. Checking safety properties using induction and a SAT solver FMCAD 2000.
24. Sheeran, M., G. Stälmarck. A tutorial on Stälmarck's method of propositional proof, Formal Methods In System Design, 16(1), 2000.
25. Silva P.M., Sakallah K., Robust search algorithms for test pattern generation, FTCS, 1997.
26. Strichman, O., Accelerating bounded model checking of safety properties, Formal Methods in System Design, vol, 24, 2004.
27. Whittemore, J., K. Kim, K. Sakallah, SATIRE: A new incremental satisfiability engine, DAC, 2001.
28. Wieringa, S. On incremental satisfiability and bounded model checking, DIFTS, 2011.