Exploiting Signal Unobservability for Efficient Translation to CNF in Formal Verification of Microprocessors

Miroslav N. Velev
mvelev@ece.cmu.edu
http://www.ece.cmu.edu/~mvelev

Abstract

The paper presents a method for translating Boolean circuits to CNF by identifying trees of ITE operators, where each ITE has fanout count of 1, and representing every such tree with a single set of equivalent CNF clauses without intermediate variables for ITE outputs, except for the tree output. This not only eliminates intermediate variables, but also reduces the number of clauses, compared to conventional translation to CNF, where each ITE is assigned an output variable and is represented with a separate set of clauses. Other gates with fanout count of 1 are similarly merged with their fanout gate to generate a single set of equivalent clauses. This translation to CNF was implemented in a decision procedure for the logic of Equality with Uninterpreted Functions and Memories (EUFM), and was applied to formulas from formal verification of microprocessors. To increase the number of ITE-trees in the Boolean formulas, the decision procedure was optimized to preserve the ITE-tree structure of arguments to equality comparisons. In conventional translation to CNF with the unoptimized decision procedure, the benchmark formulas require up to hundreds of thousands of CNF variables and millions of clauses. The best translation strategy reduced the CNF variables by up to 8×; the clauses by up to 17×; the SAT-solver decisions by up to 79×; the SAT-solver conflicts by up to 96×; and accelerated the SAT solving by up to 420×.

1. Introduction

The recent dramatic improvements in the speed of SAT-solvers [26][49][53] (see [42][68] for comparative studies) made them invaluable in formal verification. The most common input format of SAT-solvers is Conjunctive Normal Form (CNF) [31], where a Boolean formula is represented as a conjunction of *clauses*, and every clause is a disjunction of *literals*, i.e., CNF variables or their negations. In conventional translation of Boolean circuits to CNF, as first presented by Tseitin [57], a new CNF variable is introduced for the output of every logic gate, and a separate set of CNF clauses correlates that variable with the CNF variables for the inputs of the gate, given the gate's function. However, this translation captures only local structural information for each gate, and so does not exploit the concept of *observability don't-cares* [20][54] (related to the concept of dominators in testing [1])—that the output values of subcircuits may be *unobservable* at the main output, i.e., do not influence its value, given the values of other signals, and so those subcircuits can be pruned from the solution space, and their inputs left unassigned or given arbitrary values.

Gupta et al. [28] were first to implement a circuit-based SAT-solver that uses structural information to identify gates with unobservable outputs and remove the clauses for those gates, as long as the gates remain unobservable. Other circuit-based SAT-solvers identify signals with equal or complemented values in order to prune the solution space [36][45][50], or use a hybrid representation of Boolean circuits [24][50]—gate-level for the circuit, and CNF for constraints and learnt clauses. The goal in this paper is to encode more structural information about a Boolean circuit in its CNF formula, compared to conventional translation to CNF, thus allowing us to benefit from improvements in the widely researched CNF-based SAT-solvers, while exploiting the unobservability of signals to prune the solution space.

CNF-based SAT-solvers face two main hurdles to further improvements. First, the operation-intensive *Boolean Constraint Propagation* (BCP)—reflecting a CNF variable's assignment on all the clauses containing that variable or its negation—taking up to 90% of the SAT-solving time [49], and generating many non-sequential memory accesses that are prone to L2-cache misses. Furthermore, BCP requires data-dependent branches that are hard to predict, and so frequently incur the branch misprediction penalty—at least 19 cycles, and up to 125 instructions in the Intel Pentium 4 [30]. Second, many L2-cache misses for big formulas [73], resulting in expensive accesses to main memory; the L2-cache miss penalty is hundreds of cycles currently, and is increasing [30].

This paper extends the ideas from [70] (see Sect. 2.3), where Boolean circuits were translated to CNF by merging logic gates with *fanout count* of 1, i.e., whose output is used only once, with the *fanout gate* that they drive, and expressing the combined function of those gates with a single set of CNF clauses, thus eliminating the CNF variable for the output of the driving gate, and saving CNF clauses. The idea is similar to that in technology mapping—an NP-complete problem [34] that can be solved efficiently by heuristics minimizing a cost function. The translation strategies used in the current paper reduce the CNF variables and clauses, thus reducing both the required BCP and the L2-cache misses. These techniques were implemented a decision procedure for the logic of Equality with Uninterpreted Functions and Memories (EUFM) [15]. The decision procedure translates an EUFM formula to an equivalent Boolean formula that can then be checked for being a tautology with any SAT method. The CNF-translation strategies were evaluated on formulas from formal verification of microprocessors. An earlier version of this decision procedure [67] was used at Motorola [37] to formally verify a model of the M•CORE processor, and detected bugs.

The paper makes two contributions: 1) a study of strategies for translating Boolean formulas to CNF by merging trees of ITE operators; and 2) an optimization of the EUFM decision procedure by eliminating equality comparisons in a way that preserves the ITE-tree structure of the arguments.

2. Previous Translations

2.1 From EUFM to Propositional Logic

The syntax of EUFM [15] includes *terms* and *formulas*. Terms are used to abstract word-level values of data, register identifiers, memory addresses, and the entire states of memory arrays. A term can be an Uninterpreted Function (UF) applied to a list of argument terms, a term variable, or an ITE operator selecting between two argument terms based on a controlling formula, such that *ITE*(*formula*, *term1*, *term2*) will evaluate to *term1* if *formula* = *true*, and to *term2* if *formula* = *false*. The syntax for terms can be extended to model memories by means of the interpreted functions *read* and *write* [15][66] that satisfy the *forwarding property* of the memory semantics—that a *read* gets the data

1530-1591/04 $20.00 (c) 2004 IEEE

value written by the most recent *write* to the same address, or the value from the initial memory state otherwise. Formulas are used to model the control path of a microprocessor, and to express the correctness condition. A formula can be an Uninterpreted Predicate (UP) applied to a list of argument terms, a Boolean variable, an ITE operator selecting between two argument formulas based on a controlling formula, or an *equation* (equality comparison) of two terms. Formulas can be negated and combined by Boolean connectives. We will refer to both terms and formulas as *expressions*. UFs and UPs are used to abstract functional units by replacing them with "black boxes" that satisfy only the property of *functional consistency*—that equal inputs to the UF (UP) produce equal output values.

Restrictions on the style for describing high-level processors [62][63] reduced the number of terms that appear in both positive and negated equations (called *g-terms* for general terms), and increased the number of terms that appear only in positive equations (called *p-terms* for positive terms). The property of Positive Equality [62][63] allows us to treat syntactically different p-terms as not equal when evaluating the validity of an EUFM formula, thus achieving dramatic simplifications and orders of magnitude speedup (see [13] for correctness proof). However, equations between g-term variables can be either *true* or *false*, and can be encoded with Boolean variables [25][52][69], by accounting for the property of transitivity of equality [14].

Applications of the same UF or UP are eliminated with nested ITEs [63]. For example, if $p(a_1, b_1)$, $p(a_2, b_2)$, and $p(a_3, b_3)$ are three applications of UP p, where a_1, b_1, a_2, b_2, a_3, and b_3 are terms, then the first application will be eliminated with a new Boolean variable c_1, the second with $ITE((a_2 = a_1) \wedge (b_2 = b_1), c_1, c_2)$, where c_2 is a new Boolean variable, and the third with $ITE((a_3 = a_1) \wedge (b_3 = b_1), c_1, ITE((a_3 = a_2) \wedge (b_3 = b_2), c_2, c_3))$, where c_3 is a new Boolean variable. That is, the second, third, and any subsequent applications of the UP are eliminated with *ITE-chains* that enforce functional consistency. The same method for eliminating UFs and UPs is used in [38][39][55][67]. Alternatively, functional consistency can be enforced with Ackermann constraints [2]—the three applications of the UP will be replaced with the new Boolean variables c_1, c_2, and c_3; then, the functional consistency of the second application of the UP with respect to the first will be enforced by extending the resulting formula with the constraint $(a_2 = a_1) \wedge (b_2 = b_1) \Rightarrow (c_2 = c_1)$, with such constraints added for each pair of applications of that UP. This method for enforcing functional consistency is used in [6][32][43][52][58][72], but does not result in ITE-trees, and so will not benefit from the CNF translation in Sect. 3. ITE-trees also result after eliminating a *read* from a sequence of *write*s by accounting for the forwarding property of the memory semantics, and from modeling conditional instruction flow due to stalling.

After the UFs are eliminated, the terms consist of only ITE operators and term variables. In earlier EUFM decision procedures that exploit Positive Equality [38][39][55][67], equations between nested-ITE terms are eliminated by pushing the equations to the ITE leaves, and replacing the original equation with a disjunction of formulas. For example, given terms $ITE(c_1, a_1, a_2)$ and $ITE(c_2, b_1, b_2)$, where c_1 and c_2 are formulas, and a_1, a_2, b_1, and b_2 are term variables, the equation $ITE(c_1, a_1, a_2) = ITE(c_2, b_1, b_2)$ will be replaced with the formula $c_1 \wedge c_2 \wedge (a_1 = b_1) \vee c_1 \wedge \neg c_2 \wedge (a_1 = b_2) \vee \neg c_1 \wedge c_2 \wedge (a_2 = b_1) \vee \neg c_1 \wedge \neg c_2 \wedge (a_2 = b_2)$. Note the structure of the resulting formula—*a disjunction of conjunctions of formulas*, such that the ITE-tree structure of the original equation arguments is lost.

In the EUFM decision procedure used in this paper, the final Boolean formula consists of AND, OR, NOT, and ITE gates. Hashing [63] ensures that: there are no duplicate gates; merges an AND having another AND as input into a single AND, and similarly for ORs; eliminates duplicate inputs; and replaces an AND/OR with a constant if the gate has complemented inputs.

2.2 Conventional Translation from Propositional Logic to CNF

A *primary CNF variable* is one representing the value of a primary input, i.e., input of the original Boolean circuit. An *auxiliary CNF variable* is one representing the value of a gate output. In general, the translation of Boolean formulas to CNF is exponential. However, by introducing a new CNF variable for the output of every logic gate, and imposing constraints that preserve the function of that gate [57], we get a satisfiability-equivalent [17] CNF representation. Both the size of the resulting CNF and the complexity of the translation procedure are linear in the size of the original Boolean formula. For AND, OR, NOT, and ITE gates, the conventional translation to CNF is:

$o \leftarrow \text{AND}(i_1, i_2, \ldots, i_n)$:
$(i_1 \vee \neg o) \wedge (i_2 \vee \neg o) \wedge \ldots \wedge (i_n \vee \neg o) \wedge (\neg i_1 \vee \neg i_2 \vee \ldots \vee \neg i_n \vee o)$

$o \leftarrow \text{OR}(i_1, i_2, \ldots, i_n)$:
$(\neg i_1 \vee o) \wedge (\neg i_2 \vee o) \wedge \ldots \wedge (\neg i_n \vee o) \wedge (i_1 \vee i_2 \vee \ldots \vee i_n \vee \neg o)$

$o \leftarrow \text{NOT}(i)$:
$(\neg i \vee \neg o) \wedge (i \vee o)$

$o \leftarrow \text{ITE}(i, t, e)$:
$(\neg i \vee \neg t \vee o) \wedge (\neg i \vee t \vee \neg o) \wedge (i \vee \neg e \vee o) \wedge (i \vee e \vee \neg o)$

Instead of explicitly translating the inverters (NOT gates), we can subsume them in their fanout gates [51], by replacing all instances of the CNF variable for the inverter output with the negated CNF variable for the inverter input, thus eliminating the output variable and the 2 clauses for each inverter.

2.3 Translation from Propositional Logic to CNF by Merging Gates

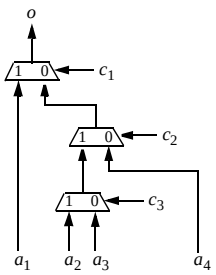
To reduce the number of CNF variables and clauses, we can merge groups of adjacent gates—where those at lower topological levels have fanout count of 1—and represent each such group with a single set of clauses [70]. This avoids the intermediate CNF variables for outputs of the gates at lower levels, and results in fewer clauses. For example, a chain of n nested ITEs—$ITE(i_1, t_1, ITE(i_2, t_2, ..., ITE(i_n, t_n, e_n))$, where the else-expression of each ITE is another ITE with fanout count of 1, and the innermost ITE has an else-expression, e_n, that is either not an ITE or has fanout count greater than 1—is translated to CNF by means of $2n + 2$ clauses: 2 for each of the $n + 1$ chain inputs e_n, and t_j ($j = 1, ..., n$). For each chain input, the first clause expresses the condition that if the input is *true* and is selected by a corresponding assignment to controlling formulas, i_j, of ITE operators that are ahead in the chain, then the chain output, o, should be *true*; the second clause expresses the condition that if the input is *false* and selected, then the output, o, should be *false*:

$(\neg i_1 \vee \neg t_1 \vee o) \wedge (\neg i_1 \vee t_1 \vee \neg o)$
$\wedge (i_1 \vee \neg i_2 \vee \neg t_2 \vee o) \wedge (i_1 \vee \neg i_2 \vee t_2 \vee \neg o) \wedge \ldots$
$\wedge (i_1 \vee i_2 \vee \ldots \vee i_n \vee \neg e_n \vee o) \wedge (i_1 \vee i_2 \vee \ldots \vee i_n \vee e_n \vee \neg o)$

Similar translations [70] can be devised for: AND/OR→ITE groups, where an ITE has an n-input AND or OR as its then-input, or else-input, or a different AND/OR gate at each of these inputs; and OR/ITE→AND (AND/ITE→OR) groups, where an n-input AND (OR) has as input an m-input OR (AND), or an ITE. Note that a driven AND (OR) gate may have many OR/ITE (AND/ITE) inputs with fanout count of 1. Then, we can choose which one to merge by using a variant of the FANIN heuristic [47] for BDD variable ordering—selecting the input gate with highest topological level. The motivation is to shorten the longest path for BCP from a primary input to the output of the driven AND (OR) gate. Thus, if the heuristic is applied to many groups, we could significantly shorten many paths for BCP from primary inputs to the output of the Boolean circuit.

3. Translation to CNF by Merging ITE-Trees

The CNF translation of ITE-chains can be generalized to ITE-trees, where the ITEs inside the tree have fanout count of 1, and each leaf is either not an ITE or an ITE with fanout count greater than 1. As for ITE-chains, for every path from a non-controlling input of the tree, we introduce 2 clauses: the first expressing the condition that if the input is *true* and is selected to appear at the tree output, *o*, by a corresponding assignment to controlling formulas of ITEs that are ahead in the tree, then the tree output should be *true*; the second expressing the condition that if the input is *false* and selected, then the tree output, *o*, should be *false*. An example ITE-tree and its translation are shown in Fig. 1.



Path from a_1 to o:
$(\neg a_1 \vee \neg c_1 \vee o) \wedge (a_1 \vee \neg c_1 \vee \neg o)$

Path from a_2 to o:
$(\neg a_2 \vee c_1 \vee \neg c_2 \vee \neg c_3 \vee o) \wedge (a_2 \vee c_1 \vee \neg c_2 \vee \neg c_3 \vee \neg o)$

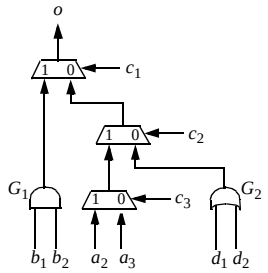
Path from a_3 to o:
$(\neg a_3 \vee c_1 \vee \neg c_2 \vee c_3 \vee o) \wedge (a_3 \vee c_1 \vee \neg c_2 \vee c_3 \vee \neg o)$

Path from a_4 to o:
$(\neg a_4 \vee c_1 \vee c_2 \vee o) \wedge (a_4 \vee c_1 \vee c_2 \vee \neg o)$

Figure 1. Example ITE-tree, and its translation to CNF with a unified set of clauses without intermediate variables for ITE outputs inside the tree. The ITE-tree is represented with the conjunction of all clauses for paths from non-controlling tree inputs to the tree output. ITEs are shown as multiplexors.

The benefits from merging ITE-trees include: **1) Reduced variables and clauses**—relative to conventional CNF translation where each ITE is translated separately—but possibly increased literals (as was the case for 2 benchmarks in the experiments in Sect. 5). **2) Reduced solution space**—the fewer CNF variables allow a SAT-solver to make fewer decisions when evaluating a formula; removing the unimportant CNF variables—representing signals inside ITE-trees—improves the efficiency of the search, allowing a SAT-solver to make decisions only based on important variables that control ITE-trees. **3) Reduced BCP**—eliminating intermediate variables for outputs of ITEs inside a tree, allows a SAT-solver to quickly propagate the value of an ITE-tree input to the tree output; if the literals increase, the BCP will also increase, but the benefits from reduced variables and clauses more than offset this in the experiments. **4) Automatic use of signal unobservability**—the 2 clauses, introduced for each path in an ITE-tree, become satisfied as soon as an ITE-controlling signal selects another path, allowing a SAT-solver more freedom in assigning values to unobservable signals. **5) Reduced L2-cache misses**—the fewer variables and clauses result in smaller CNF file sizes, and more succinctly represent the solution space, allowing the clauses for the active portion of the search to better fit in the L2 cache; also, the average number of literals per clause increases, and since the literals for a clause are situated in contiguous memory locations, SAT-solvers can better exploit the spacial locality of memory accesses [30].

The ITE-trees can be further merged with their AND/OR leaves with fanout count of 1—see Fig. 2. Each of the clauses from conventional translation of those AND/OR gates (see Sect. 2.2) is extended with the condition that the gate output is either not selected for the tree output, or the tree output has the same value as the gate output, so that the literal for the gate output becomes unnecessary and is deleted. This saves 1 variable for the gate output, and 2 clauses for the tree path from the gate output to the tree output, but adds literals if that path is longer than 4 ITEs.



Path from G_1 to o:
$(\neg b_1 \vee \neg b_2 \vee \neg c_1 \vee o) \wedge (b_1 \vee \neg c_1 \vee \neg o) \wedge (b_2 \vee \neg c_1 \vee \neg o)$

Path from a_2 to o:
$(\neg a_2 \vee c_1 \vee \neg c_2 \vee \neg c_3 \vee o) \wedge (a_2 \vee c_1 \vee \neg c_2 \vee \neg c_3 \vee \neg o)$

Path from a_3 to o:
$(\neg a_3 \vee c_1 \vee \neg c_2 \vee c_3 \vee o) \wedge (a_3 \vee c_1 \vee \neg c_2 \vee c_3 \vee \neg o)$

Path from G_2 to o:
$(\neg d_1 \vee c_1 \vee c_2 \vee o) \wedge (\neg d_2 \vee c_1 \vee c_2 \vee o) \wedge (d_1 \vee d_1 \vee c_1 \vee c_2 \vee \neg o)$

Figure 2. Merging an ITE-tree with its AND/OR leaves that have fanout count of 1.

When merging ITE-trees, two questions are: 1) whether an ITE-tree should include ITEs with fanout count greater than 1, or have such ITEs only as leaves; and 2) if merging the AND/OR leaves that have fanout count of 1, whether to also merge AND/OR leaves with fanout count greater than 1. These will be answered in Sect. 5, based on experimental results.

4. Preserving the ITE-Tree Structure of Equation Arguments in Translation from EUFM

Instead of eliminating equations with disjunctions of conjunctions (Sect. 2.1), we can eliminate them by pushing the equations to the term-variable leaves of the nested-ITE arguments, and preserving the ITE-tree structure. For example, the equation $ITE(c_1, a_1, a_2) = ITE(c_2, b_1, b_2)$, will be transformed to $ITE(c_1, ITE(c_2, a_1 = b_1, a_1 = b_2), ITE(c_2, a_2 = b_1, a_2 = b_2))$. Then, equations with the same term variable as arguments are replaced with *true*; those between two syntactically different p-term variables are replaced with *false*, based on Positive Equality; and those between two syntactically different g-term variables can be encoded with a new Boolean variable [25]. Since in equations between p-terms, most of the resulting equations between term variables will be between syntactically different term variables, and so evaluate to *false*, a hybrid scheme was used for p-term equations—pushing the equation to the leaves of only one of the arguments, keeping the other intact, and then using the original method to produce a disjunction of conjunctions. Although pushing both arguments to their leaves and then simplifying may slightly increase the resulting ITE-trees, the hybrid scheme was much faster for formulas with big p-terms. One question is: in what order to push the equation arguments to their leaves. Of several studied approaches, two had advantage—1) first the argument with fewer topological levels; and 2) first the one with more levels—such that running the EUFM decision procedure with both, and then SAT-solving the formula with fewer clauses was consistently best. An alternative scheme that, for each equation, estimates the clauses resulting from each order of pushing the two arguments to their leaves, and then greedily chooses the order that adds fewer clauses, actually produced CNF formulas that had more total clauses and took longer SAT times, compared to the better of the above two strategies, due to not accounting for how ITE-trees share their inputs.

5. Results

The Boolean formulas used in the experiments are from formal verification of safety of the benchmarks: ooo_engine6, an out-of-order processor with a 6-entry reorder buffer, 6 reservation stations, register renaming, and register-register ALU instructions; 1dlx_iq50, a single-issue pipelined DLX [30], modeled as in [63], and extended with a 50-entry instruction queue between the instruction memory and the execution pipeline; 1dlx_m_iq52, an implementation with multicycle functional units, exceptions, and branch prediction, modeled as in [64], and having a 52-entry instruction queue; 9vliw_iq2, a 9-wide VLIW processor with predicated execution, register remapping, advanced loads (see [65]), and a 2-entry instruction queue; 9vliw_iq6, a variant with a 6-entry instruction queue; 14pipe, a 14-wide, 5-stage pipelined processor with in-order execution, implementing register-register ALU and load instructions [68]; and 8pipe_ooo, an 8-wide, 5-stage pipelined processor with out-of-order execution, also implementing register-register ALU and load instructions [68].

Table 1 summarizes the results. The abstraction function [15] for the verification was computed by controlled flushing [16], where the user provides a flushing schedule that eliminates the triggering of stalling conditions, thus simplifying the correctness formula. The equations between g-term variables were encoded with e_{ij} Boolean variables [24]—new Boolean variables, each representing the outcome of a unique equation between different g-term variables. The e_{ij} encoding outperformed two other encoding methods [52][69], when the translation techniques from Sections 3 and 4 were applied. Transitivity of equality [14] was enforced when formally verifying 10 buggy variants of 9vliw_iq6, as well as the correct versions of the 2 processors with out-of-order execution, ooo_engine6 and 8pipe_ooo; the other 5 processors have in-order execution and do not require transitivity, so that this property was not enforced for their correct versions. The SAT-solver Siege_v3 [53]—one of the winners in the SAT'03 competition [42]—was used for the experiments. The computer was a Dell OptiPlex GX260 with a 3.06-GHz Intel Pentium 4, having a 512-KB on-chip L2-cache, 2 GB of memory, and running Red Hat Linux 9.

As Table 1 shows, the formulas had between 335 and 21,330 Boolean variables before translation to CNF. With the original method of eliminating equations in an EUFM decision procedure (see Sect. 2.1) and conventional translation to CNF without subsuming the inverters, i.e., Strategy 0, the CNF variables were between 37,600 and 506,408; the CNF clauses between 551,775 and 17,597,056; Siege_v3 made between 0.89×10^6 and $2{,}609.11\times10^6$ decisions, resolved between 0.71×10^6 and 92.10×10^6 conflicts, and took between 1,553 and 377,224 seconds to prove the CNF formulas unsatisfiable.

The results from seven other strategies are also included in Table 1. Strategy "old best"—subsuming inverters, merging ITE-chains, and merging AND/OR→ITE groups—is based on the translations in Sect. 2.3, and had best performance in [70]. In the rest of the strategies, the techniques from Sect. 4 were used to preserve the ITE-tree structure of equation arguments ("equations-to-ITEs"). The first 3 strategies employ merging of ITE-trees without merging their AND/OR leaves: Strategy 1, merging ITE-trees and AND/OR→ITE groups; Strategy 2, also merging ITE→AND and ITE→OR groups, such that if there are several inputs that are ITEs with fanout count of 1, the ITE with highest topological level is chosen, based on a variant of the FANIN heuristic [47] (see Sect 2.3); Strategy 3, merging OR/ITE→AND and AND/ITE→OR groups, also based on the FANIN heuristic, such that input gates that are AND/OR were selected only if they have fewer inputs than a parameter `INPUT_LIMIT`. Strategies 4, 5, and 6 are analogous to Strategies 1, 2, and 3, respectively, except that ITE-trees are merged with their leaves that are AND/OR gates with fanout count of 1. For strategies 3 and 6, parameter `INPUT_LIMIT` was empirically found to be 4, i.e., the input OR (AND) could have either 2 or 3 inputs. With each of the 6 new strategies, the time for translation to CNF was almost identical to that with conventional translation, and sometimes less.

Without the optimization of the EUFM decision procedure to preserve the ITE-tree structure of equation arguments, merging ITE-trees and other gate groups, as otherwise done in Strategies 1–6, resulted in approximately the same number of ITE-trees as there are ITE-chains detected by Strategy "old best", so that the CNF formulas had similar statistics and required similar SAT times as the CNF formulas produced by Strategy "old best".

From Table 1, Strategies 1–6 resulted in Boolean correctness formulas with 5.25–14.7× more ITEs, compared to Strategies 0 and "old best" that employ the previous method for eliminating equations in the EUFM decision procedure. This is due to preserving the ITE-tree structure of equation arguments (see Sect. 4), and correspondingly decreases the AND, OR, and NOT gates that would have otherwise be used to represent the eliminated equations as disjunctions of conjunctions of formulas. Thus, the increase in the number of ITE-trees by 2.5–51×, with the average depth of ITE-trees more than doubling for most of the benchmarks, and the maximum depth usually increasing by 2–13×.

Strategies 1–3 have almost identical CNF statistics, and so do Strategies 4–6 (see Table 1). Compared to Strategy 0, Strategies 1–3 reduced the CNF variables by 3–8×, the clauses by 3–17×, the literals by 2–7.7× (with the exception of 1dlx_iq50 and 1dlx_m_iq52, where the literals increased by 1.24× and 1.10×, respectively), and the CNF file sizes by 2–9× (again, with the exception of 1dlx_iq50 and 1dlx_m_iq52, where the reduction was small). Strategies 1–3 resulted in similar SAT-checking times. However, *Strategy 2 had the best performance on 5 of the 7 benchmarks*—1dlx_c_iq50, 1dlx_m_iq52, 9vliw_iq2, 14pipe, and 8pipe_ooo—and reduced the SAT-solving decisions by 1.5–78.7×, the conflicts by 2.5–97×, and sped up the SAT-solving by 4.4–426.2×. Although Strategies 4–6 further reduced the CNF variables by 11–21%, compared to Strategies 1–3, the reduction in clauses was insignificant, while the literals increased by up to 2×, thus increasing the BCP and L2-cache misses, and usually slowing the SAT-solver. The numbers of CNF variables, clauses, and literals are not proportional to the SAT time with Strategies 0 and "old best", but are closely correlated with Strategies 1–6.

We next answer the two questions posed at the end of Sect. 3: 1) extending Strategy 1 to merge ITE-trees with ITE leaves that have fanout count greater than 1, increased the clauses by up to 2×, the literals by up to 3×, and the SAT time by up to 4×; and 2) extending Strategy 4 to merge ITE-trees with their AND/OR leaves that have fanout count greater than 1, increased the clauses by up to 4×, the literals by up to 5×, and the SAT time by up to 40×. Hence, ITE-trees should not be merged with ITE and AND/OR leaves with fanout count greater than 1.

Strategy 4 had the best performance among the 6 new strategies, when formally verifying 10 buggy variants of 9vliw_iq6 that result in satisfiable CNF formulas. The better performance relative to Strategies 1–3 can be explained with the shortened paths from primary inputs to the output of the Boolean circuit for the correctness condition, due to merging of ITE-trees with their AND/OR leaves with fanout count of 1. Using the extended version of Strategy 4, also merging ITE-trees with their AND/OR leaves with fanout count greater than 1—in order to shorten more paths from primary inputs to the Boolean circuit output—resulted in equal or better performance for 5 of the 10 benchmarks. Strategy "old best" was better than Strategy 4 on 4 of the buggy models, and Strategy 0 was better than Strategy 4 on 5 models, thus the potential for further speedups with parallel runs of the tool flow, each with different strategy, as used in [56][68].

Strategies 1–6 resulted in similar speedups, for both satisfiable and unsatisfiable formulas, with another efficient SAT-solver, BerkMin621 [27], a new version of BerkMin62 [26].

Table 1. Results from unsatisfiable formulas. Speedup is the ratio between the SAT time with the previous translation from EUFM to propositional logic followed by conventional translation to CNF (i.e., Strategy 0), and the new SAT time.

Boolean Formula (Boolean Variables)	Strategy for Translation from EUFM to CNF	Gates in the Boolean Formula				ITE-trees			CNF Formula					SAT-solver Siege_v3			Speedup
		ITEs	ANDs	ORs	NOTs	#	Avg. Depth	Max. Depth	Vars	Clauses	Literals	Avg. Literals/ Clause	Size [MB]	Decisions $\times 10^6$	Conflicts $\times 10^6$	Time [sec]	
ooo_engine6 (335)	(0): conventional translation to CNF	6,282	36,780	2,038	2,037	——	——	——	47,440	614,098	1,760,646	2.867	11.7	0.89	0.71	1,553	——
	(old best): subsume NOTs; merge ITE-chains, AND/OR→ITE	6,282	36,780	2,038	2,037	1,074	3.92	5	40,239	599,696	1,762,097	2.938	11.4	0.62	0.49	810	1.9×
	(1): equations-to-ITEs; merge ITE-trees, AND/OR→ITE	38,277	9,692	529	1,782	4,924	5.93	37	16,140	186,772	964,613	5.165	5.5	0.30	0.23	319	4.9×
	(2): (1) + FANIN for ITE→AND, ITE→OR	38,277	9,692	529	1,782	4,924	5.93	37	15,653	185,798	964,193	5.189	5.5	0.33	0.26	355	4.4×
	(3): (2) + FANIN for OR/ITE→AND, AND/ITE→OR	38,277	9,692	529	1,782	4,924	5.93	37	15,511	185,514	964,263	5.198	5.5	0.30	0.23	313	**5.0**×
	(4): (1) + merge ITE-trees with AND/OR leaves	38,277	9,692	529	1,782	4,924	5.93	37	12,774	180,040	1,141,583	6.341	6.2	0.37	0.29	381	4.1×
	(5): (4) + FANIN for ITE→AND, ITE→OR	38,277	9,692	529	1,782	4,924	5.93	37	12,452	179,396	1,141,583	6.363	6.1	0.31	0.24	312	**5.0**×
	(6): (5) + FANIN for OR/ITE→AND, AND/ITE→OR	38,277	9,692	529	1,782	4,924	5.93	37	12,310	179,112	1,141,653	6.374	6.1	0.34	0.27	366	4.2×
1dlx_iq50 (3,819)	(0): conventional translation to CNF	98,697	292,445	10,579	10,995	——	——	——	416,535	5,914,135	17,114,365	2.894	135.1	36.05	0.72	8,240	——
	(old best): subsume NOTs; merge ITE-chains, AND/OR→ITE	98,697	292,445	10,579	10,995	5,254	16.95	51	311,910	5,704,885	18,969,171	3.325	145.7	21.74	0.60	3,829	2.2×
	(1): equations-to-ITEs; merge ITE-trees, AND/OR→ITE	534,585	51,091	364	9,609	24,723	14.63	101	79,708	2,029,326	21,234,409	10.464	131.7	19.09	0.27	1,065	7.7×
	(2): (1) + FANIN for ITE→AND, ITE→OR	534,585	51,091	364	9,609	24,723	14.63	101	79,646	2,029,202	21,234,233	10.464	131.7	20.24	0.28	980	**8.4**×
	(3): (2) + FANIN for OR/ITE→AND, AND/ITE→OR	534,585	51,091	364	9,609	24,723	14.63	101	79,592	2,029,094	21,234,174	10.464	131.7	23.48	0.38	1,434	5.7×
	(4): (1) + merge ITE-trees with AND/OR leaves	534,585	51,091	364	9,609	24,723	14.63	101	68,057	2,006,024	27,539,672	13.728	166.3	24.67	0.30	1,079	7.6×
	(5): (4) + FANIN for ITE→AND, ITE→OR	534,585	51,091	364	9,609	24,723	14.63	101	67,995	2,005,900	27,539,496	13.729	166.3	31.00	0.36	1,329	6.2×
	(6): (5) + FANIN for OR/ITE→AND, AND/ITE→OR	534,585	51,091	364	9,609	24,723	14.63	101	67,941	2,005,792	27,539,437	13.730	166.2	24.98	0.31	1,203	6.8×
1dlx_m_iq52 (10,702)	(0): conventional translation to CNF	118,545	351,679	12,178	13,304	——	——	——	506,408	7,908,816	22,972,124	2.905	183.7	63.02	2.54	26,916	——
	(old best): subsume NOTs; merge ITE-chains, AND/OR→ITE	118,545	351,679	12,178	13,304	6,165	16.65	107	373,004	7,642,008	25,288,928	3.309	197.2	56.48	3.40	20,653	1.3×
	(1): equations-to-ITEs; merge ITE-trees, AND/OR→ITE	623,391	71,469	1,226	11,532	30,894	13.62	116	107,622	2,456,168	25,463,080	10.367	160.2	44.74	1.36	4,563	5.9×
	(2): (1) + FANIN for ITE→AND, ITE→OR	623,391	71,469	1,226	11,532	30,894	13.62	116	107,454	2,455,832	25,462,585	10.368	160.2	41.30	1.06	3,762	**7.2**×
	(3): (2) + FANIN for OR/ITE→AND, AND/ITE→OR	623,391	71,469	1,226	11,532	30,894	13.62	116	107,327	2,455,578	25,462,452	10.369	160.1	50.52	1.65	5,567	4.8×
	(4): (1) + merge ITE-trees with AND/OR leaves	623,391	71,469	1,226	11,532	30,894	13.62	116	94,548	2,430,020	32,843,573	13.516	201.3	45.90	1.46	4,687	5.7×
	(5): (4) + FANIN for ITE→AND, ITE→OR	623,391	71,469	1,226	11,532	30,894	13.62	116	94,435	2,429,794	32,843,243	13.517	201.3	50.41	1.46	5,285	5.1×
	(6): (5) + FANIN for OR/ITE→AND, AND/ITE→OR	623,391	71,469	1,226	11,532	30,894	13.62	116	94,308	2,429,540	32,843,110	13.518	201.2	51.47	1.55	5,366	5.0×
9vliw_iq2 (6,402)	(0): conventional translation to CNF	1,444	32,138	4,111	4,761	——	——	——	48,856	551,775	1,573,303	2.851	10.7	27.25	4.72	3,195	——
	(old best): subsume NOTs; merge ITE-chains, AND/OR→ITE	1,444	32,138	4,111	4,761	115	2.10	3	42,755	539,573	1,561,013	2.893	10.5	9.68	0.99	579	5.5×
	(1): equations-to-ITEs; merge ITE-trees, AND/OR→ITE	21,294	11,233	76	2,725	3,558	4.71	39	22,143	177,462	796,088	4.486	4.9	9.13	0.79	282	11.3×
	(2): (1) + FANIN for ITE→AND, ITE→OR	21,294	11,233	76	2,725	3,558	4.71	39	21,947	177,070	795,739	4.494	4.9	6.01	0.40	141	**22.7**×
	(3): (2) + FANIN for OR/ITE→AND, AND/ITE→OR	21,294	11,233	76	2,725	3,558	4.71	39	21,918	177,012	795,732	4.495	4.9	8.82	0.69	250	12.8×
	(4): (1) + merge ITE-trees with AND/OR leaves	21,294	11,233	76	2,725	3,558	4.71	39	19,502	172,180	1,321,565	7.675	7.6	7.78	0.71	230	13.9×
	(5): (4) + FANIN for ITE→AND, ITE→OR	21,294	11,233	76	2,725	3,558	4.71	39	19,307	171,790	1,321,219	7.691	7.6	9.71	0.95	318	10.1×
	(6): (5) + FANIN for OR/ITE→AND, AND/ITE→OR	21,294	11,233	76	2,725	3,558	4.71	39	19,282	171,740	1,321,218	7.693	7.6	8.22	0.80	254	12.6×
9vliw_iq6 (21,330)	(0): conventional translation to CNF	10,360	165,464	12,570	14,005	——	——	——	223,729	3,662,687	10,603,981	2.895	79.5	1,454.97	52.88	141,857	——
	(old best): subsume NOTs; merge ITE-chains, AND/OR→ITE	10,360	165,464	12,570	14,005	335	2.53	7	199,340	3,613,909	10,595,485	2.932	78.6	286.45	8.74	18,480	7.7×
	(1): equations-to-ITEs; merge ITE-trees, AND/OR→ITE	126,386	42,259	123	8,077	17,225	5.62	71	82,685	852,932	4,879,393	5.721	31.6	198.14	3.12	5,468	25.9×
	(2): (1) + FANIN for ITE→AND, ITE→OR	126,386	42,259	123	8,077	17,225	5.62	71	82,373	852,308	4,878,920	5.724	31.6	197.88	3.18	5,892	24.1×
	(3): (2) + FANIN for OR/ITE→AND, AND/ITE→OR	126,386	42,259	123	8,077	17,225	5.62	71	82,323	852,208	4,878,867	5.725	31.6	139.05	2.21	3,841	**36.9**×
	(4): (1) + merge ITE-trees with AND/OR leaves	126,386	42,259	123	8,077	17,225	5.62	71	73,293	834,148	8,047,690	9.648	49.5	160.03	2.35	3,856	36.8×
	(5): (4) + FANIN for ITE→AND, ITE→OR	126,386	42,259	123	8,077	17,225	5.62	71	72,982	833,526	8,047,220	9.654	49.5	180.96	2.68	4,407	32.2×
	(6): (5) + FANIN for OR/ITE→AND, AND/ITE→OR	126,386	42,259	123	8,077	17,225	5.62	71	72,936	833,434	8,047,173	9.655	49.5	200.43	2.96	5,329	26.6×
14pipe (3,848)	(0): conventional translation to CNF	16,100	161,823	3,207	6,069	——	——	——	184,980	17,597,056	52,461,115	2.981	347.1	2,609.11	92.10	377,224	——
	(old best): subsume NOTs; merge ITE-chains, AND/OR→ITE	16,100	161,823	3,207	6,069	1,978	7.93	14	171,036	17,569,171	52,565,271	2.992	343.1	922.03	31.33	110,347	3.4×
	(1): equations-to-ITEs; merge ITE-trees, AND/OR→ITE	103,312	15,081	288	3,899	5,096	18.23	82	22,657	1,020,587	6,749,832	6.614	38.7	33.15	0.95	945	399.2×
	(2): (1) + FANIN for ITE→AND, ITE→OR	103,312	15,081	288	3,899	5,096	18.23	82	22,657	1,020,587	6,749,832	6.614	38.7	33.15	0.95	885	**426.2**×
	(3): (2) + FANIN for OR/ITE→AND, AND/ITE→OR	103,312	15,081	288	3,899	5,096	18.23	82	22,369	1,020,011	6,749,464	6.617	38.4	37.21	1.05	998	378.0×
	(4): (1) + merge ITE-trees with AND/OR leaves	103,312	15,081	288	3,899	5,096	18.23	82	19,253	1,013,779	12,050,385	11.887	66.2	46.82	1.44	1,449	260.3×
	(5): (4) + FANIN for ITE→AND, ITE→OR	103,312	15,081	288	3,899	5,096	18.23	82	19,253	1,013,779	12,050,385	11.887	66.2	46.82	1.44	1,559	242.0×
	(6): (5) + FANIN for OR/ITE→AND, AND/ITE→OR	103,312	15,081	288	3,899	5,096	18.23	82	18,965	1,013,203	12,050,017	11.893	65.9	51.53	1.55	1,590	237.2×
8pipe_ooo (2,209)	(0): conventional translation to CNF	2,960	29,171	1,170	2,090	——	——	——	37,600	1,195,395	3,521,321	2.946	23.2	57.49	28.52	47,020	——
	(old best): subsume NOTs; merge ITE-chains, AND/OR→ITE	2,960	29,171	1,170	2,090	562	4.94	8	33,164	1,186,523	3,519,527	2.966	23.0	32.75	16.59	22,322	2.1×
	(1): equations-to-ITEs; merge ITE-trees, AND/OR→ITE	19,718	5,096	177	1,378	1,585	10.67	39	8,502	251,444	1,146,716	4.561	6.6	1.51	0.40	202	232.8×
	(2): (1) + FANIN for ITE→AND, ITE→OR	19,718	5,096	177	1,378	1,585	10.67	39	8,500	251,440	1,146,722	4.561	6.6	1.61	0.42	196	**239.9**×
	(3): (2) + FANIN for OR/ITE→AND, AND/ITE→OR	19,718	5,096	177	1,378	1,585	10.67	39	8,351	251,142	1,146,474	4.565	6.5	1.46	0.40	198	237.5×
	(4): (1) + merge ITE-trees with AND/OR leaves	19,718	5,096	177	1,378	1,585	10.67	39	7,413	249,266	1,351,621	5.422	7.6	1.68	0.44	217	216.7×
	(5): (4) + FANIN for ITE→AND, ITE→OR	19,718	5,096	177	1,378	1,585	10.67	39	7,413	249,266	1,351,621	5.422	7.6	1.68	0.44	216	217.7×
	(6): (5) + FANIN for OR/ITE→AND, AND/ITE→OR	19,718	5,096	177	1,378	1,585	10.67	39	7,264	248,968	1,351,373	5,428	7.6	1.71	0.46	226	208.1×

6. Related Work

Variations of Tseitin's transformations [57] were used in [7][9] [21][22][33][51], and Larrabee [40] was first to apply them to testing of circuits, but none of these authors used transformations for ITE-trees. Kuehlmann et al. [36] represented Boolean circuits in terms of 2-input AND gates and inverters, and transformed groups of 3 connected AND gates, one driven by the other two, into a canonical form by accounting for inverters at gate inputs.

Franco et al. [23] present a circuit-based SAT-solver that uses BDDs [12] in its decision heuristic, and was faster than Chaff [49] on formulas from Bounded Model Checking, but slower on formulas from EUFM-based formal verification of processors that are much simpler than those used here. Theoretical results about circuit-based SAT algorithms are presented in [3][11].

Burch [16] extended SVC [32], a decision procedure for EUFM, with simplifications that are based on observability don't-cares, but his method depended on manually provided case-splitting expressions, and was computationally expensive even for relatively simple benchmarks.

To reduce the cost of BCP, Bingham and Hu [8] compiled Boolean formulas to programs, and simulated the resulting code with random vectors. Additional code was generated to identify input patterns that will produce the same output value as the current vector, thus pruning the solution space.

Algebraic simplifications for CNF [5][10][19][41][44][46] [48] require long processing times for big formulas. Three of those methods [10][46][48], as well as techniques for CNF variable ordering by minimizing the cut-width [4][71] (see also [18]), were applied to simpler formulas from formal verification of processors [68], but took long time, and did not accelerate the SAT checking. Using multiple parallel runs of a SAT-solver with either different decision heuristics [56], or with different translations from EUFM to propositional logic [68], and stopping as soon as one of the runs finds an answer, reduced the SAT time.

Nested ITEs were first used to eliminate UFs in [61], where bit-level functional units were abstracted with read-only instances of an Efficient Memory Model (EMM) [59][60] for behavioral abstraction of memory arrays in symbolic simulation.

The EMM has been adopted in verification tools by Innologic Systems [29], and Synopsys [35].

7. Conclusions

By preserving the ITE-tree structure of equation arguments, when translating EUFM formulas to equivalent Boolean formulas, and then by merging the ITE-trees and representing each of them with a unified set of clauses, we can reduce the CNF variables by up to 8×, the clauses by up to 17×, and achieve speedup of up to 420×. When verifying correct processors, resulting in unsatisfiable CNF formulas, best was Strategy 2—subsuming inverters, merging ITE-trees, merging AND/OR→ITE groups, and using a variant of the FANIN heuristic to merge ITE→AND and ITE→OR groups. For buggy processors, resulting in satisfiable CNF formulas, best was Strategy 4—merging the ITE-trees with their AND/OR leaves with fanout count of 1, and merging AND/OR→ITE groups. The significant speedup was possible due to using the nested-ITE scheme for eliminating uninterpreted functions and uninterpreted predicates in the EUFM formulas, thus producing terms that consist of ITE-trees. Future work will fine-tune the presented strategies.

References

[1] M. Abramovici, M.A. Breuer, and A.D. Friedman, *Digital Systems Testing and Testable Design*, revised edition, IEEE Press, New York, 1990.

[2] W. Ackermann, *Solvable Cases of the Decision Problem*, North-Holland, Amsterdam, 1954.

[3] M. Alekhnovich, and A.A. Razborov, "Satisfiability, Branch-Width and Tseitin Tautologies," *Foundations of Computer Science (FOCS '02)*, November 2002, pp. 593–603.

[4] F.A. Aloul, I.L. Markov, and K.A. Sakallah, "Faster SAT and Smaller BDDs via Common Function Structure," *International Conference on Computer-Aided Design (ICCAD '01)*, November 2001, pp. 443–448.

[5] F. Bacchus, and J. Winter, "Effective Preprocessing with Hyper-Resolution and Equality Reduction," *6th International Conference on Theory and Applications of Satisfiability Testing (SAT '03)*, May 2003.

[6] C. Barrett, D. Dill, and A. Stump, "Checking Satisfiability of First-Order Formulas by Incremental Translation to SAT," *Computer-Aided Verification (CAV '02)*, LNCS 2404, Springer-Verlag, July 2002, pp. 236–249.

[7] M. Bauer, D. Brand, M. Fischer, A. Meyer, and M. Paterson, "A Note on Disjunctive Form Tautologies," SIGACT News, Vol. 4 (1973), pp. 17–20.

[8] J.D. Bingham, and A.J. Hu, "Semi-Formal Bounded Model Checking," *Computer-Aided Verification (CAV '02)*, LNCS 2404, Springer-Verlag, July 2002.

[9] T. Boy de la Tour, "An Optimality Result for Clause Form Translation," Journal of Symbolic Computation, Vol. 14 (1992), pp. 283–301.

[10] R.I. Brafman, "A Simplifier for Propositional Formulas with Many Binary Clauses," *Int'l. Joint Conference on Artificial Intelligence (IJCAI '01)*, 2001.

[11] E. Broering, and S.V. Lokam, "Width-Based Algorithms for SAT and CIRCUIT-SAT," *6th Int'l. Conference on Theory and Applications of Satisfiability Testing (SAT '03)*, May 2003.

[12] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," IEEE Transactions on Computers, Vol. C-35, No. 8 (August 1986), pp. 677–691.

[13] R.E. Bryant, S. German, and M.N. Velev, "Processor Verification Using Efficient Reductions of the Logic of Uninterpreted Functions to Propositional Logic," ACM Transactions on Computational Logic (TOCL), Vol. 2, No. 1 (January 2001), pp. 93–134.

[14] R.E. Bryant, and M.N. Velev, "Boolean Satisfiability with Transitivity Constraints," ACM Transactions on Computational Logic (TOCL), Vol. 3, No. 4 (October 2002), pp. 604–627.

[15] J.R. Burch, and D.L. Dill, "Automated Verification of Pipelined Microprocessor Control," *Computer-Aided Verification (CAV '94)*, LNCS 818, Springer-Verlag, June 1994.

[16] J.R. Burch, "Techniques for Verifying Superscalar Microprocessors," *33rd Design Automation Conference (DAC '96)*, June 1996, pp. 552–557.

[17] H.K. Büning, and T. Lettmann, *Propositional Logic: Deduction and Algorithms*, Cambridge Tracts in Theoretical Computer Science 48, Cambridge University Press, 1999.

[18] E.M. Clarke, and O. Strichman, "A Failed Attempt to Optimize Variable Ordering with Tools for Constraints Solving," *Workshop on Constraints in Formal Verification (CFV '02)*, 2002.

[19] J.M. Crawford, and L.D. Auton, "Experimental Results on the Crossover Point in Satisfiability Problems," *11th National Conference on Artificial Intelligence*, 1993, pp. 21–27. http://citeseer.nj.nec.com/crawford93experimental.html

[20] M. Damiani, and G. De Micheli, "Observability Don't Care Sets and Boolean Relations," *International Conference on Computer-Aided Design (ICCAD '90)*, November 1990.

[21] E. Eder, "An Implementation of a Theorem Prover Based on the Connection Method," *Artificial Intelligence: Methodology, Systems, Applications (AIMSA '84)*, 1985.

[22] U. Egly, and T. Rath, "On the Practical Value of Different Definitional Translations to Normal Form," *13th International Conference on Automated Deduction (CADE '96)*, M. McRobbie, and J. Slaney, *eds.*, LNCS 1104, July–August 1996.

[23] J. Franco, M. Kouril, J. Schlipf, J. Ward, S. Weaver, M. Dransfield, and W.M. Vanfleet, "SBSAT: A State-Based, BDD-Based Satisfiability Solver," *6th International Conference on Theory and Applications of Satisfiability Testing (SAT '03)*, May 2003.

[24] M.K. Ganai, L. Zhang, P. Ashar, A. Gupta, S. Malik, "Combining Strengths of Circuit-Based and CNF-Based Algorithms for a High-Performance SAT Solver," *39th Design Automation Conference (DAC '02)*, June 2002.

[25] A. Goel, K. Sajid, H. Zhou, A. Aziz, and V. Singhal, "BDD Based Procedures for a Theory of Equality with Uninterpreted Functions," *Computer-Aided Verification (CAV '98)*, LNCS 1427, Springer-Verlag, June 1998, pp. 244–255.

[26] E. Goldberg, and Y. Novikov, "BerkMin: A Fast and Robust Sat-Solver," *Design, Automation, and Test in Europe (DATE '02)*, March 2002, pp. 142–149.

[27] E. Goldberg, and Y. Novikov, Personal Communication, June 2003.

[28] A. Gupta, A. Gupta, Z. Yang, and P. Ashar, "Dynamic Detection and Removal of Inactive Clauses in SAT with Application in Image Computation," *38th Design Automation Conference (DAC '01)*, June 2001, pp. 536–541.

[29] G. Hasteer, Personal Communication, February 1999.

[30] J.L. Hennessy, and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd edition, Morgan Kaufmann, San Francisco, 2002.

[31] D.S. Johnson, and M.A. Trick, *eds.*, *The Second DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. http://dimacs.rutgers.edu/challenges

[32] R.B. Jones, D.L. Dill, and J.R. Burch, "Efficient Validity Checking for Processor Verification," *International Conference on Computer-Aided Design (ICCAD '95)*, 1995.

[33] T.A. Junttila, and I. Niemelä, "Towards and Efficient Tableau Method for Boolean Circuit Satisfiability Checking," *1st International Conference on Computational Logic (CL '00)*, LNAI 1861, Springer-Verlag, July 2000.

[34] K. Keutzer, "DAGON: Technology Binding and Local Optimization by DAG Matching," *24th Design Automation Conference (DAC '87)*, June 1987.

[35] A. Kölbl, J.H. Kukula, K. Antreich, and R.F. Damiano, "Handling Special Constructs in Symbolic Simulation," *39th Design Automation Conference (DAC '02)*, June 2002.

[36] A. Kuehlmann, M.K. Ganai, and V. Paruthi, "Circuit-Based Boolean Reasoning," *38th Design Automation Conference (DAC '01)*, June 2001, pp. 232–237.

[37] S. Lahiri, C. Pixley, and K. Albin, "Experience with Term Level Modeling and Verification of the M•CORE™ Microprocessor Core," *International Workshop on High Level Design, Validation and Test (HLDVT '01)*, November 2001.

[38] S.K. Lahiri, S.A. Seshia, and R.E. Bryant, "Modeling and Verification of Out-of-Order Microprocessors in UCLID," *Formal Methods in Computer-Aided Design (FMCAD '02)*, LNCS 2517, Springer-Verlag, November 2002.

[39] S.K. Lahiri, and R.E. Bryant, "Deductive Verification of Advanced Out-of-Order Microprocessors," *Computer-Aided Verification (CAV '03)*, LNCS, Springer-Verlag, July 2003.

[40] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Vol. 11, No. 1 (January 1992), pp. 4–15.

[41] D. Le Berre, "Exploiting the Real Power of Unit Propagation Lookahead," *Workshop on Theory and Applications of Satisfiability Testing (SAT '01)*, H. Kautz, and B. Selman, *eds.*, Elsevier Science Publishers, Electronic Notes in Discrete Mathematics, Vol. 9, June 2001.

[42] D. Le Berre, and L. Simon, "Results from the SAT'03 Solver Competition," *6th International Conference on Theory and Applications of Satisfiability Testing (SAT '03)*, 2003. http://www.lri.fr/~simon/contest03/results/

[43] J. Levitt, and K. Olukotun, "Verifying Correct Pipeline Implementation for Microprocessors," *International Conference on Computer-Aided Design (ICCAD '97)*, 1997.

[44] C.M. Li, and Anbulagan, "Look-Ahead versus Look-Back for Satisfiability Problems," *3rd International Conference on Principles and Practice of Constraint Programming (CP '97)*, Springer-Verlag, LNCS 1330, 1997, pp. 342–356.

[45] F. Lu, L.-C. Wang, K.-T. Cheng, J. Moondanos, and Z. Hanna, "A Signal Correlation Guided ATPG Solver and Its Applications for Solving Difficult Industrial Cases," *40th Design Automation Conference (DAC '03)*, June 2003.

[46] I. Lynce, and J.P. Marques-Silva, "Probing-Based Preprocessing Techniques for Propositional Satisfiability," *15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '03)*, November 2003.

[47] S. Malik, A.R. Wang, R.K. Brayton, and A. Sangiovani-Vincentelli, "Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment," *International Conference on Computer-Aided Design (ICCAD '88)*, November 1988.

[48] J.P. Marques-Silva, "Algebraic Simplification Techniques for Propositional Satisfiability," *Principles and Practice of Constraint Programming (CP '00)*, Rina Dechter, *ed.*, LNCS 1894, Springer-Verlag, September 2000.

[49] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," *38th Design Automation Conference (DAC '01)*, June 2001.

[50] R. Ostrowski, E. Grégoire, B. Mazure, and L. Saïs, "Recovering and Exploiting Structural Knowledge from CNF Formulas," *Principles and Practice of Constraint Programming (CP '02)*, P. Van Hentenryck, *ed.*, LNCS 2470, Springer-Verlag, September 2002, pp. 185–199.

[51] D.A. Plaisted, and S. Greenbaum, "A Structure Preserving Clause Form Translation," Journal of Symbolic Computation (JSC), Vol. 2, 1985, pp. 293–304.

[52] A. Pnueli, Y. Rodeh, O. Strichman, and M. Siegel, "The Small Model Property: How Small Can It Be?", Journal of Information and Computation, Vol. 178, No. 1 (October 2002).

[53] L. Ryan, Siege SAT Solver v.3. http://www.cs.sfu.ca/~loryan/personal/

[54] H. Savoj, and R.K. Brayton, "The Use of Observability and External Don't Cares for the Simplification of Multi-Level Networks," *Design Automation Conference (DAC '00)*, June 1990.

[55] S.A. Seshia, S.K. Lahiri, and R.E. Bryant, "A Hybrid SAT-Based Decision Procedure for Separation Logic with Uninterpreted Functions," *Design Automation Conference (DAC '03)*, June 2003, pp. 425–430.

[56] O. Shacham, and E. Zarpas, "Tuning the VSIDS Decision Heuristic for Bounded Model Checking," *Microprocessor Test and Verification (MTV '03)*, May 2003.

[57] G.S. Tseitin, "On the Complexity of Derivation in Propositional Calculus," in *Studies in Constructive Mathematics and Mathematical Logic*, Part 2, 1968, pp. 115–125. Reprinted in J. Siekmann, and G. Wrightson, *eds.*, *Automation of Reasoning*, Vol. 2, Springer-Verlag, 1983.

[58] O. Tveretina, and H. Zantema, "A Proof System and a Decision Procedure for Equality Logic," Technical Report, Department of Computer Science, Technical University of Eindhoven, 2003. http://www.win.tue.nl/~hzantema/TZ.pdf

[59] M.N. Velev, and R.E. Bryant, "Efficient Modeling of Memory Arrays in Symbolic Ternary Simulation," *Tools and Algorithms for the Construction and Analysis of Systems (TACAS '98)*, B. Steffen, *ed.*, LNCS 1384, Springer-Verlag, March–April 1998.

[60] M.N. Velev, and R.E. Bryant, "Incorporating Timing Constraints in the Efficient Memory Model for Symbolic Ternary Simulation," *International Conference on Computer Design (ICCD '98)*, October 1998, pp. 400–406.

[61] M.N. Velev, and R.E. Bryant, "Bit-Level Abstraction in the Verification of Pipelined Microprocessors by Correspondence Checking," *Formal Methods in Computer-Aided Design (FMCAD '98)*, LNCS 1522, Springer-Verlag, November 1998, pp. 18–35.

[62] M.N. Velev, and R.E. Bryant, "Exploiting Positive Equality and Partial Non-Consistency in the Formal Verification of Pipelined Microprocessors," *36th Design Automation Conference (DAC '99)*, June 1999, pp. 397–401.

[63] M.N. Velev, and R.E. Bryant, "Superscalar Processor Verification Using Efficient Reductions of the Logic of Equality with Uninterpreted Functions to Propositional Logic," *Correct Hardware Design and Verification Methods (CHARME '99)*, LNCS 1703, Springer-Verlag, September 1999, pp. 37–53.

[64] M.N. Velev, and R.E. Bryant, "Formal Verification of Superscalar Microprocessors with Multicycle Functional Units, Exceptions, and Branch Prediction," *37th Design Automation Conference (DAC '00)*, June 2000, pp. 112–117.

[65] M.N. Velev, "Formal Verification of VLIW Microprocessors with Speculative Execution," *Computer-Aided Verification (CAV '00)*, LNCS 1855, Springer-Verlag, July 2000.

[66] M.N. Velev, "Automatic Abstraction of Memories in the Formal Verification of Superscalar Microprocessors," *Tools and Algorithms for the Construction and Analysis of Systems (TACAS '01)*, T. Margaria, and W. Yi, *eds.*, LNCS 2031, Springer-Verlag, April 2001, pp. 252–267.

[67] M.N. Velev, and R.E. Bryant, "EVC: A Validity Checker for the Logic of Equality with Uninterpreted Functions and Memories, Exploiting Positive Equality and Conservative Transformations," *Computer-Aided Verification (CAV '01)*, LNCS 2102, Springer-Verlag, July 2001.

[68] M.N. Velev, and R.E. Bryant, "Effective Use of Boolean Satisfiability Procedures in the Formal Verification of Superscalar and VLIW Microprocessors," Journal of Symbolic Computation (JSC), Vol. 35, No. 2 (February 2003), pp. 73–106.

[69] M.N. Velev, "Automatic Abstraction of Equations in a Logic of Equality," *Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX '03)*, M.C. Mayer, and F. Pirri, *eds.*, LNAI 2796, Springer-Verlag, September 2003, pp. 189–206.

[70] M.N. Velev, "Efficient Translation of Boolean Formulas to CNF in Formal Verification of Microprocessors," *Asia and South Pacific Design Automation Conference (ASP-DAC '04)*, January 2004.

[71] D. Wang, E. Clarke, Y. Zhu, and J. Kukula, "Using Cutwidth to Improve Symbolic Simulation and Boolean Satisfiability," *IEEE International High Level Design Validation and Test Workshop (HLDVT '01)*, 2001.

[72] H. Zantema, and J.F. Groote, "Transforming Equality Logic to Propositional Logic," *4th International Workshop on First Order Theorem Proving (FTP '03)*, June 2003. http://www.win.tue.nl/~jfg/articles/CS-Report03-05.pdf

[73] L. Zhang, and S. Malik, "Cache Performance of SAT Solvers: A Case Study for Efficient Implementation of Algorithms," *6th International Conference on Theory and Applications of Satisfiability Testing (SAT '03)*, May 2003.