Considering Circuit Observability Don't Cares in CNF Satisfiability*

Zhaohui Fu† Yinlei Yu Sharad Malik
Department of Electrical Engineering
Princeton University
Princeton, NJ 08544, USA
{zfu, yyu, sharad}@Princeton.EDU

ABSTRACT

Boolean Satisfiability (SAT) has seen significant use in various tasks in circuit verification in recent years. A key contributor to the efficiency of contemporary SAT solvers is fast deduction using Boolean Constraint Propagation (BCP). This can be efficiently implemented with a Conjunctive Normal Form (CNF) representation of a circuit. However, most circuit verification tasks start from a logic circuit description of the problem instance. Fortunately, there is a simple conversion from a logic circuit to a CNF [12] that enables the use of the CNF representation even for circuit verification tasks. However, this process loses some information regarding the structure of the circuit. One example of such structural information is the Circuit Observability Don't Cares. Several recent papers [6] [7] [8] [9] [11] [13] have addressed the issue of handling circuit unobservability in CNF-based SAT. However, as we will demonstrate, none of these accurately captures the conditions for use of this information in all stages of a CNF-based SAT solver. In this paper, we propose a broader approach to take such Don't Care information into consideration in a CNF-based SAT solver. It does so by adding certain *don't care literals* to clauses in the CNF representation. These don't care literals are treated differently at different times during the solution process, much like don't cares in logic synthesis. The major merit of this scheme, unlike other recently proposed techniques, is that the solver can continue to use this don't care information during the learning process, which is an important part of contemporary SAT solvers. We have implemented this approach in the zChaff SAT solver and experiments show that significant performance gain can be obtained through their use.

1. INTRODUCTION

Boolean Satisfiability (SAT) is probably one of the most well studied combinatorial optimization problems. Researchers have devoted significant effort to developing efficient SAT solvers. For decades, Electronic Design Automation (EDA), and in particular synthesis and verification, has been one of the major drivers for SAT research.

Most SAT solvers work on a CNF representation of a formula. The primary motivation is that the iterative application of the unit clause rule (referred to as Boolean Constraint Propagation or BCP) can be done very efficiently using this data structure, and in fact is the workhorse of almost all SAT solvers. This is not a limitation for using SAT on formulas that arise from circuits as there is a simple conversion for capturing a circuit as a CNF. This involves taking the conjunction of the consistency conditions for each gate in the circuit expressed in CNF form. This is linear in the size of the circuit. However, the disadvantage of using a CNF SAT solver over a circuit based verification technique is the lack of circuit structural information. Such structural information that includes the direction of gates and Circuit Observability Don't Cares (Cir-ODCs), is lost after the circuit is translated into a CNF formula. This information is potentially useful in the solution process. This paper attempts to bridge this gap by providing an approach that takes Cir-ODCs into account during the SAT solving process.

The proposed approach identifies the Cir-ODCs in a given circuit statically, i.e. prior to the start of the SAT solving process. This information is passed to the SAT solver, which in turn adjusts its decision heuristic, BCP and conflict driven procedures based on this additional information. *Most importantly, our approach distinguishes itself from other methods of handling observability don't cares by propagating this don't care information during the learning process that is an integral part of modern SAT solvers.* This ability is extremely useful because the number of clauses that the SAT solver *learns* during the search for a solution is usually much larger than the number of original clauses.

While in general, the unobservability condition for a signal is represented as a set of cubes, we (and all other competing approaches) limit ourselves to literals for efficient use in SAT solvers. We also consider an extension of the static scheme presented in the paper that can potentially exploit a larger set of don't cares by allocating them dynamically during the SAT solving process rather than statically. However, in practice, the overhead of managing them generally overweighs their performance advantage.

This paper is organized as follows. Section 2 describes how Cir-ODCs arise in circuits. Our static approach is discussed in detail in Section 3. Section 4 describes the modification of a CNF SAT solver for handling Cir-ODCs. Next, we very briefly explain the dynamic approach for handling Cir-ODCs in Section 5. Experimental results are presented and discussed in Section 6. We give a review of related work in Section 7 and then provide conclusions and future directions in Section 8.

2. CIRCUIT OBSERVABILITY DON'T CARES

The notion of circuit observability don't cares is as follows. Some signal *s*, under certain conditions *C*, no longer affects the outputs of the circuit. These conditions, *C*, are referred to as observability don't care conditions for *s*.

We consider circuits with simple gates (`AND`, `OR`, `NOT`, `NOR`, `NAND`) as these are typically used as the starting point in either

*This work is supported by grant 2002-TJ-1025 from SRC Corporation and grant 2001-111DT from Intel Corporation.

†Z. Fu is also a Senior Tutor at the Department of Computer Science, National University of Singapore, Singapore 117543.

CNF or circuit based SAT solvers due to the ease of deduction.

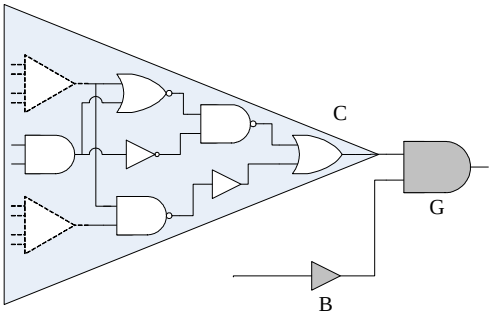

Figure 1: An AND gate G takes as input the only fan-out from a logic cone C.

Consider Figure 1 with a simple two input AND gate G, which takes the output value of a logic block, or a logic "cone", C of gates as one of its inputs. The output value of the AND gate G is fixed to be 0 if its input from B is zero, regardless of the output value of C, and hence the entire logic cone of C need not be considered any more if it has only one fan-out that goes into G. Correspondingly in SAT, all the clauses corresponding to C can be ignored in the SAT solver in subsequent search. These clauses are labeled as inactive in [7]. We treat these clauses as Cir-ODC clauses and those gate outputs as Cir-ODC variables, since they are unobservable at the outputs of the circuit. Note that the Cir-ODC status of a clause or a variable is a dynamic property in SAT search since it depends on the current assignment during the search.

Ignoring the Cir-ODC clauses in CNF will not change the satisfiability of the original problem. However, it potentially leaves the variables in these clauses under-constrained. This is not an issue if the instance is unsatisfiable, since even with being under-constrained the instance has already been shown to be unsatisfiable. Consider Figure 1 again. When $B_{out} = 0$, the output of G is 0 regardless of the output value of C. If the original problem is unsatisfiable, the unsatisfiability must not involve any clauses in C. This is because the output of C is no longer being constrained when $B_{out} = 0$. However, if the instance is satisfiable, then to get a complete assignment, we need to ensure that the final assignment of variables in the Cir-ODC clauses is consistent with the circuit. This is easily accomplished by no longer ignoring the Cir-ODCs once the instance is determined to be SAT. Consider the following Algorithm 1, where v is a variable and V is the set of variables in the CNF.

Algorithm 1 Re-validate the truth assignment.

```
1: for all variables v ∈ V(CNF) do
2:     if v only appears in ignored Cir-ODC clauses then
3:         Unassign the truth value of v
4:     end if
5: end for
6: Re-solve the CNF instance with the current partial assignment
```

The main idea behind Algorithm 1 is that the only inconsistency in the satisfying assignment of the CNF formula comes from the clauses that are completely ignored due to Cir-ODC.

3. A STATIC MECHANISM FOR HANDLING CIR-ODCS IN CNF-SAT

We start with a basic review of the use of Cir-ODCs in SAT solvers. Consider a portion of a typical logic circuit in Figure 2. The lower case letters are the outputs associated with each gate and they also correspond to the variables in the CNF formula of the circuit. The value of a is unobservable if either b or c is 0 since the output f is determined independent of a. We denote that $\{\bar{b}, \bar{c}\}$ (i.e. $\bar{b}+\bar{c}$) is the *Cir-ODC condition* to make a unobservable. Thus, if we propagate this Cir-ODC condition to the gates G_2 and G_3, all clauses associated with these gates become Cir-ODC when this condition is true, i.e. either b or c is set to 0. The backward propagation of Cir-ODC condition only stops at some gate that has fan-out outside the transitive fan-in cone of f.

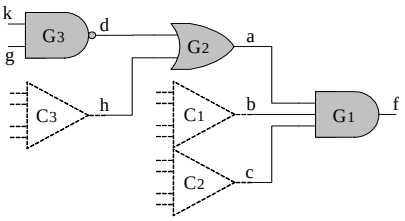

Figure 2: Cir-ODC condition associated with a three input AND gate.

3.1 Total Ordering of Gate Inputs

Consider the following case in Figure 2. First a is set to 0, making both b and c unobservable. Then c is set to 0. Setting c to 0 by itself makes a and b unobservable. There is a problem in considering both a and c as unobservable as at least one of a and c needs to take responsibility for setting f to 0. This can be taken care of by a total ordering on the inputs of a, b and c, i.e. only a lower order signal can appear in the Cir-ODC condition of a higher order one. This issue is discussed in the context of using ODCs in logic synthesis [1]. In fact, any arbitrary ordering π of gate inputs is sufficient to ensure that one gate input is responsible for generating the controlling value at the gate output. For a two input logic AND gate with inputs u and v, we force that $\bar{v}$ can appear in the Cir-ODC condition of u if and only if $\pi(v) < \pi(u)$. We can use the circuit in Figure 2 as an example and order the inputs alphabetically, which results in $\pi(a) < \pi(\underline{b}) < \pi(c)$. The Cir-ODC conditions for C_1 and C_2 are $\{\bar{a}\}$ and $\{\bar{a}, b\}$ respectively. Gate G_2 has an empty set as its Cir-ODC condition. Clearly, such a static scheme loses some of the Cir-ODC condition at each gate. This is because in a static total ordering of a logic AND gate with inputs u and v, either $\bar{u}$ can make v unobservable, or $\bar{v}$ can make u unobservable, but not both. Thus if we choose either one, say $\bar{u}$ is the Cir-ODC condition of v, and in the SAT solving process there is a chance that $v = 0$ is assigned earlier, which can make u unobservable, but the SAT solver cannot use that since such condition was thrown away in the translation phase just to ensure the total ordering.

3.2 Propagation of the Cir-ODC Conditions

As we have mentioned in the previous section, each gate input has a set of don't care literals that captures the condition for it to be unobservable. Each gate also inherits a set of Cir-ODC conditions from its successors. *Recall that a Cir-ODC condition is a set of don't care literals*. These Cir-ODC conditions are then propagated towards the primary inputs. A Depth First Search (DFS) from the primary inputs is sufficient for such propagation in the circuit. We start by initializing the set of don't care literals, i.e. the Cir-ODC condition, to be empty for each gate. Then we perform a DFS from

each of the inputs. The exact recursive DFS procedures are given in Algorithm 2 and 3.

Algorithm 2 Cir-ODC-Gate (Gate g)

```
1: if g is a primary output then
2:    Return ∅;
3: end if
4: C = Universal Set;
5: for all fan out edge [g,h] of g do
6:    C := C ⋂ (Cir-ODC-Gate(h) ⋃ Cir-ODC-Edge([g,h]));
7: end for
8: Return C;
```

Algorithm 3 Cir-ODC-Edge (Edge $[g,h]$)

```
1: C = ∅;
2: for all fan in gate i of h, i ≠ g do
3:    if π(i) < π(h) then
4:       C := C ⋃ {i = controlling value for h}
5:    end if
6: end for
7: Return C;
```

We illustrate the Cir-ODC condition propagation using the circuit in Figure 3. Assuming an alphabetical ordering of the inputs, i.e. $\pi(r) < \pi(s) < \pi(t)$ and $\pi(p) < \pi(q)$, all logic gates in cone C_1 inherit a Cir-ODC condition $\{\overline{r}, \overline{p}\}$. Similarly, all gates in cone C_2 inherit $\{\overline{r}, \overline{s}, \overline{p}\}$. Gate G_2 has two successors, one in C_1 and the other in C_2. The Cir-ODC condition that G_2 inherits is the intersection of the two Cir-ODC conditions from them, which is $\{\overline{r}, \overline{p}\}$, i.e. a normal set intersection.

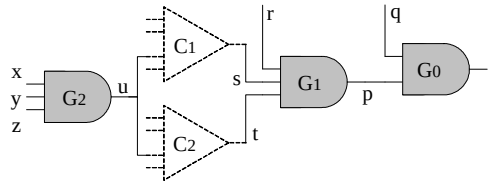

Figure 3: Two inputs of gate G_1 converge at G_2, thus G_2 has two Cir-ODC condition sets coming from the same successor, G_1.

3.3 A Greedy Ordering Heuristic for Gate Inputs

As we have discussed earlier in Section 3.1, a total order of the gate inputs is needed to ensure that at least one input will provide the controlling value of the gate when needed. For each gate, any given total order of the inputs completely determines the total number of don't care literals contributed by this gate. In order to utilize these don't care literals in CNF-SAT, we are interested in finding a good total order such that it gives the most don't care literals in the circuit. This is because a more "popular" don't care literal is likely to cause a larger portion of the circuit to be unobservable when it is set to be true.

We propose a greedy ordering heuristic that works in a similar manner as the propagation of don't care literals discussed in previous section. For each gate that could generate new don't care literals, the maximal set of don't care literals for each input is propagated towards the primary input, disregarding the total order constraint, i.e. ignoring the $\pi(i) < \pi(h)$ condition in Algorithm 3. Consider Figure 3 as an example. Don't care set $\{\overline{s}, \overline{t}, \overline{p}\}$ is propagated through input pin r; don't care set $\{\overline{r}, \overline{t}, \overline{p}\}$ is propagated through input pin s; and so on. Obviously, this violates the total ordering constraint, but such propagation gives an estimate of the number of gates that each don't care literal can eventually reach. After the propagation is finished, the gate input whose don't care literal that reaches more gates is assigned a lower order rank. A tie in the order is broken by random. When the circuit is to be translated in to CNF format, the don't care literals for each clause are also ordered in the same way as above, i.e. a don't care literal with larger use appears earlier in the clause. This helps the SAT solver pick the more important literals when it cannot accept all of them because of efficiency reasons.

4. SOLVING SAT WITH CIR-ODCS

There are several Cir-ODC related modifications that have to be made in a CNF-based SAT solver. These modifications include devising an augmented CNF format for Cir-ODC encoding, changing the BCP process to ignore the inactive Cir-ODC clauses, handling the don't care literals in conflict analysis and conflict driven clause learning and using Cir-ODC condition to guide the decision heuristics.

4.1 Format of Cir-ODC-CNF

In a CNF file using DIMACS format [5], a clause is represented with a line of literals that ends in a `0`. To minimize the change in the CNF format, we define our new file format for Cir-ODC-CNF such that each clause also has one line. This can be represented using a *regular expression* as follows.

$$\text{Clause} = l^+\ 0\ d^*\ 0$$

where l stands for a normal literal with phase $(+/-)$, this is the same as in conventional CNF. l^+ means that there are 1 or more normal literals. d is a don't care literal and d^* means that there may be 0 or more don't care literals. `0`s are used as delimiters and a line starting with a leading `0` ends the input. A clause can then be ignored if any of the don't care literals is set to be true. Consider the example in Figure 3 and G_2 has a Cir-ODC condition (don't care set) of $\{\overline{r}, \overline{p}\}$. Suppose the variable indices are $p = 1, r = 2, s = 3, t = 4, x = 5, y = 6, z = 7$ and $u = 8$. Also suppose p occurs more frequently as a don't care literal in the circuit. The following four lines will be generated in the Cir-ODC-CNF file.

```
5 -8 0 -1 -2 0
6 -8 0 -1 -2 0
7 -8 0 -1 -2 0
-5 -6 -7 8 0 -1 -2 0
```

Note that in the above example, p and r appear as don't care literals in these clauses. They may appear as regular, i.e. not don't care, literals in other clauses.

4.2 Enabling zChaff with Cir-ODCs

BCP in zChaff and other state-of-the-art CNF SAT solvers benefits significantly from the two literal watching scheme [10]. We will assume familiarity with this and will not be reviewing this for sake of brevity.

With the augmented clause description, the don't care literals are not watched but examined during two literal watching. If a don't care literal is satisfied, the clause is considered to be satisfied. This terminates the search for a new literal to watch and thus can potentially terminate BCP quicker. Essentially the don't care literals

have a conditional don't care status. They can only satisfy a clause early without needing to be zero before an implication is derived from the clause. This is completely different from Velev's use of unobservability variables [13], in which the don't care literals can no longer be differentiated from the regular ones after the CNF translation.

The checking of don't care literals in a clause is performed in a *lazy* way. The clauses are not marked inactive or ignored until they are accessed during BCP. This is an advantage over the other approaches of explicitly marking clauses active/inactive, which need an explicit bookkeeping overhead.

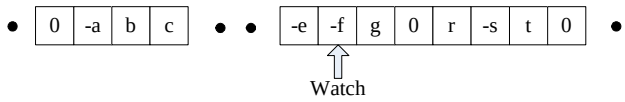

Figure 4: A typical clause in zChaff's literal pool using the static approach.

Consider the example given in Figure 4. The clause shown has literals $\bar{a}, b, c, \ldots, \bar{e}, \bar{f}, g$ with don't care literals $r, \bar{s}, t$ delimited by a `0`. Suppose one of the two watch pointers is initially pointing to literal $\bar{f}$. When $f = 1$, this clause is examined in BCP since $\bar{f} = 0$ and we need to find a new unassigned literal to watch. The watch pointer then moves to the next literal after $\bar{f}$, which is g. If g is assigned to be 0, the watch pointer continues moving right and it reaches the don't care delimiter `0`. The watch pointer continues to move right and examines the don't care literals. If any of the don't care literals is found to be true, e.g. $\bar{s} = 1$, the clause can be ignored and the BCP for this clause is immediately terminated without scanning the rest of the clause, i.e. $\bar{a}, b, c, \ldots, \bar{e}$. Thus, if a clause is ignored (inactive), there will be no implications based on this clause. Note that even though the don't care literals are directly integrated into zChaff's literal pool, they must be differentiated from other normal literals since a don't care literal is never allowed to be watched. In other words, there should be no implication on a don't care literal. When a clause is ignored, we leave the watch pointer unchanged from its position before the clause was ignored. Note that this is slightly different from the original zChaff implementation of the two literal watching scheme. We use Figure 4 as an example. In the ordinary two literal watching scheme, the watch pointer needs to be updated such that it points to the recently discovered true literal, in this example, $\bar{s}$. However in our case, $\bar{s}$ cannot be watched since it is a don't care literal. We leave the watch pointer on $\bar{f}$ unchanged. This modification does ensure that the original invariant (i.e. each "not yet satisfied" clause has two watch pointers pointing to two different unassigned literals) holds after a backtrack. This is because $dlevel(s) \leq dlevel(\bar{f})$[1], i.e. $\bar{s} = 1$ is assigned at a decision level no later than $\bar{f} = 0$. This clause can only resume its "not yet satisfied" status on a backtrack to a decision level $d < dlevel(s)$, by which time $\bar{f}$ has already been unassigned since $d < dlevel(s) \leq dlevel(\bar{f})$.

zChaff uses the Variable State Independent Decaying Sum (VSIDS) [10] decision heuristic. VSIDS is based on the number of occurrences of a literal in the CNF, known as literal count, and this is updated periodically. For each don't care literal in the clause, we also increase its literal count. This has a direct impact on the decision heuristic, where a frequently occurring don't care literal may be chosen and assigned to its controlling value. This, in turn, sets clauses to be ignored. The original SAT problem is thus potentially simplified due to the decrease in the total number of unsatisfied clauses. We believe that the performance gain for the static approach comes partially from the increasing scores of the don't care literals, as they are more likely to be branched on earlier in the search.

Handling this Cir-ODC condition in conflict clause generation is the main contributor in the performance gain, as we will demonstrate in Section 6. Recall that learned clauses are particularly useful in generating implications (i.e. forced assignments) during the search process. However, we would like these implications to be useful, i.e. arise from observable active parts of the circuits. The following algorithm provides for this. The normal literals in the learned conflict clause are obtained in the same way as before using resolution on a set of clauses. The don't care literals of the learned clause are obtained as follows. A set union operation of all the don't care literals from the clauses involved in the resolution captures all the don't care literals for the learned clause. The rationale for this is as follows. The learned clause is satisfied if any of its source clauses is satisfied. These source clauses are satisfied when any of the don't care literals is true. Thus the don't care literal set of the learned clause is the set union of the don't care literals of the source clauses. Consider the following example of resolving two clauses $(a + \bar{b} + c)$ and $(b + d + \bar{e})$ with sets of don't cares $\{r, \bar{s}\}$ and $\{\bar{x}, r, s\}$ respectively. The resolved (learned) clause is $(a + c + d + \bar{e})$ with the set of don't cares $\{r, \bar{s}, \bar{x}, s\}$. It is worth mentioning that in classical SAT if a clause contains a variable in both phases, this clause is a tautology and can be deleted from the formula. However, this is not true for a clause with don't care literals. A variable can appear in both phases in the don't care literals, as either of them may trigger the unobservability of this clause.

For implementation efficiency, when a learned clause is added to the clause database, the don't care literals are ordered according to the non-increasing order of their occurrence count and only a certain amount (proportional to the number of its normal literals) of the don't care literals are added to the learned clause.

Conflict driven clause learning with observability don't care information is a major contribution of our approach. This is not handled by either the circuit based approaches of marking inactive sections of the circuits or clauses (e.g. Gupta *et al.* [7]) or Velev's unobservability variables. In our approach, the observability don't care information can be *learned* during the searching process. This is significantly different from any other method, which can only make limited use of Cir-ODCs information provided with the original circuit. Learning of the Cir-ODC condition is very important because the majority of the clauses that the SAT solver deals with during the search process are learned conflict clauses; this is especially true for the very hard instances. Learning with Cir-ODC also contributes to the VSIDS scoring as certain key don't care literals will appear in more clauses, which results in an increase in their VSIDS scores.

5. A DYNAMIC APPROACH FOR CIR-ODC IN SAT

The static approach enforces the input ordering during the translation phase. This unavoidably reduces the total amount of Cir-ODC conditions that a CNF SAT solver can utilize. To overcome this we also considered a more dynamic technique for a SAT solver to encode the Cir-ODC conditions. In the dynamic approach, the assignment of variables during the search process is used to order the inputs of a gate. The inputs are ordered in increasing order of their being set to controlling values. Now the Cir-ODC condition of a logic gate is encoded using a set of tuples instead of just lit-

[1] $dlevel(v)$ refers to the decision level in the search tree that variable v is assigned a value.

erals. These tuples capture the dynamic ordering information. For the example in Figure 2 the Cir-ODC condition for all clauses in the transitive fan-in of signal a is if $b = 0$ or $c = 0$ *before* a is set to be 0. This is represented by the tuple $(\{\bar{a}\}, \{\bar{b}, \bar{c}\})$ in each of the these clauses.

Though the rationale behind Cir-ODC condition propagation and conflict driven clause learning in the dynamic approach is same as the one used in the static approach, the implementation is much more complicated than the static case because of the use of tuple representations. These details are omitted here for sake of brevity. As a result of more complicated operations, it is empirically observed that the overhead of the dynamic approach outweighs any benefit received from them.

6. EXPERIMENTS

In order to perform the experiments with Cir-ODCs, we need to have the combinational circuit description for the benchmarks. The only two benchmark families we found with such descriptions are `iscas85` [3] and `itc99` [4]. All test case used whose names start with a `c` are from the `iscas85` family, where `c1908` is generated from a ECAT circuit; `c2670`, `c3540`, `c5315` and `c7552` are all generated from ALU and control circuits; and `c6288` is taken from a 16-bit multiplier circuit. We use five test cases from the `itc99` family, namely `b14` from a subset of Viper processor circuit; `b15` from a subset of 80386 processor circuit; `b20`, `b21` and `b22` are composed of multiple copies of circuits that are similar to `b14`. `b14-opt` is the optimized circuit for `b14`, and similarly for `b15-opt`, `b20-opt`, `b21-opt` and `b22-opt`.

Table 1: Total and average number of don't care literals captured by random ordering and greedy ordering heuristic

bench mark	# of gates	# of clauses	Random		Heuristic	
			# of dc	dc / cls	# of dc	dc / cls
c1908	1819	4882	8K	1.53	17K	3.46
c2670	2608	6859	14K	2.10	134K	19.49
c3540	3411	9327	9K	1.00	44K	4.77
c5315	4916	14002	91K	6.50	389K	27.78
c6288	4897	14593	4K	0.29	77K	5.25
c7552	7338	19848	121K	6.11	600K	30.22
b14	20057	58594	1490K	25.44	3422K	58.39
b14-opt	11217	35510	732K	20.64	1787K	50.33
b15	17669	53468	2277K	42.58	4670K	85.46
b15-opt	14979	47862	1948K	40.69	3880K	81.06
b20	40399	118351	7903K	66.78	13801K	116.61
b20-opt	24949	79389	4690K	59.08	7929K	99.88
b21	41089	120601	7955K	65.96	14136K	117.21
b21-opt	25303	80157	4678K	58.36	8131K	101.44
b22	59849	175632	19059K	108.52	29452K	167.69
b22-opt	36119	114122	10274K	90.03	15777K	138.25

For each test case in the benchmark, we generate a miter circuit [2] that consists of two identical copies of the test circuit. A miter circuit outputs 0 if the two test circuits always output the same values and it outputs 1 otherwise. Table 1 shows the number of don't care literals using both the random ordering and the greedy ordering heuristic. Row 1 of this table corresponds to the miter version of `c1908` with 1819 logic gates. There are 4,882 CNF clauses generated using this circuit. If we use a random total ordering of the gate inputs, a total of about eight thousand don't care literals are captured, which gives an average of 1.53 don't care literals per CNF clause generated. However, if we use the greedy ordering heuristic to order the gate inputs, we could identify a total of seventeen thousand don't care literals and this gives an average of 3.46 don't care literals per CNF clause generated. Clearly, the greedy ordering heuristic is able to capture many more don't care literals than the random ordering. Besides, the greedy ordering heuristic could also order the don't care literals in a single CNF clause according to their importance, i.e. the most frequently appearing literals are placed in front of the others.

Given the numbers in Table 1, we can see that the total number of don't care literals in some circuits is extremely large. Scanning all these literals could be slow. Instead of using all the Cir-ODC condition, we restrict the number of don't care literals for each clause to not exceed a certain threshold, which is usually set to be half of the total number of normal literals in this clause.

All the experiments are conducted on a Dell PowerEdge 700 running Linux Fedora Core 1.0 (g++ GCC 3.3.2) with single Pentium 4 2.8GHz, 1MB L2 cache CPU on 800MHz main bus. Table 2 tabulates the running time of both random and heuristic ordering in the static approach (columns labeled Static Random and Static Heuristic). In addition, to study the benefit of don't care literals through the learning process, we consider the case when their use is turned off during learning (column labeled Static NL). The running time of the dynamic (column labeled Dynamic) approach and the latest zChaff is also presented for comparison.

Table 2: Running time (seconds) of the static with random and heuristic ordering, static without using Cir-ODCs in learning (Static NL) and dynamic approaches compared to zChaff.

bench mark	zChaff	Static Ordering			Dynamic Ordering
		Heuristic	Random	NL	
c1908	0.72	0.78	0.78	0.49	1.95
c2670	0.56	0.49	0.51	1.07	0.63
c3540	18.75	11.21	10.45	17.42	15.63
c5315	6.36	4.96	5.51	6.78	7.79
c7552	23.52	12.63	13.41	18.95	25.76
b14	6620.57	4079.24	3989.55	6037.52	10622.50
b14-opt	4602.08	3745.72	4005.39	4683.84	4735.81
b15	152.39	81.20	129.08	150.24	171.61
b15-opt	161.14	114.02	159.81	160.28	171.69
b20	20225.20	11014.80	17810.40	25187.20	$>$ 24 hours
b20-opt	33067.40	26662.70	28580.20	32039.30	$>$ 24 hours
b21	19625.40	10220.60	17168.30	19034.60	$>$ 24 hours
b21-opt	34711.90	12109.10	28137.70	33093.90	$>$ 24 hours
b22	22625.00	15074.50	15085.00	22032.40	$>$ 24 hours
b22-opt	34224.90	17811.10	28579.70	34948.20	$>$ 24 hours

Clearly, the static approach outperforms both the dynamic approach and zChaff. The static approach without don't care literal learning performs only slightly better (sometimes worse, as for `c2670`) than original zChaff. The main performance gain for the static method comes from the conflict clause learning with Cir-ODC. This is particularly true for large and hard benchmarks as they tend to have a large clause pool, most of which are learned conflict clauses. Don't care literals in these large number of clauses increase the VSIDS score dramatically as well as provide benefit during BCP. The dynamic approach seems to suffer from the huge overhead introduced by the tuple representation and consequent bookkeeping operations.

7. RELATED WORK

The concept of circuit observability don't cares was first introduced by Bartlett *et al.* in their work on multilevel logic minimization, where a Boolean network is optimized into a prime, irredundant R-minimal form [1].

Various circuit-based SAT solvers have been implemented. Safarpour *et al.* have dealt with both Observability Don't Cares and Controllability Don't Cares using both a CNF and circuit based

SAT solver. A variable, which is also an input to a logic gate, is marked *lazy* [11] if it is being dominated by others. They then guide the decision heuristic of the SAT solver not to branch on the lazy variables. The *laziness* then propagates through the circuit. They sometimes also mark an assigned variable lazy to erase some old bad decisions. In addition to the CNF formula, their SAT solver also needs circuit layout information in order to propagate the laziness of variables. The non-branching on lazy variables is a decision heuristic that uses some Cir-ODC information, but in a limited way, since those Cir-ODC related clauses are still active in the sense that they are generating implications, most of which are on lazy variables. Gupta *et al.* [7] implemented a CNF-based SAT solver that identifies and masks unobservable gates during SAT search. A circuit-based modification [6] of the zChaff SAT solver to avoid this directly works on the logic level representation of the circuit, it only uses CNF for the learned clauses. Thus, this technique is a hybrid CNF and circuit based SAT solver. Other more circuit-biased SAT solvers also monitor the circuit signals and utilize the correlation of different signals to prune the search space [8] [9].

Velev uses a different approach, which focuses on the translation from circuits to CNF [12] and is thus the closest to our approach. This is illustrated by the circuit given in Figure 1. Velev's approach will add an unobservability literal u to every CNF clause generated by the logic block C. Since the value of C is unobservable at the primary outputs if the output x of gate B is 0, Velev's approach adds a new clause $(x+u)$ during the CNF translation. This clause implies u to be 1 when $x=0$, which makes all clauses generated by C satisfied. Another clause $(\bar{x}+\bar{u})$ is also added to ensure the observability of the logic block C, where $x=1$ implies $u=0$ and the output of C is observable at gate G. However, it is not clear how the mutual exclusion in exploiting the don't cares is avoided for the multiple inputs of a gate. Also, once the circuit has been translated into CNF, it is impossible to differentiate the unobservability literals from normal literals. Thus all literals are treated equally and the unobservability literals will also be required to generate implications. Velev's approach is also limited since it only considers the Cir-ODC condition from the logic blocks with only one fan-out. In the experiments, the unobservability variable is only introduced for ITE-trees [13] with fan-out count of 1. Velev's approach increases the size of the translated CNF file, but does not change its format. So any stand-alone SAT solver could still be used. However, such an implementation may significantly increase both the number of variables and the number of clauses, which add to the burden of the SAT solver.

8. CONCLUSIONS AND FUTURE DIRECTIONS

In this paper we describe a technique for handling logic circuit unobservability conditions in a CNF SAT solver. The key contribution is that the don't care conditions are captured as don't care literals in the clauses describing the logic circuit. We show how the don't care information is captured from the local unobservability conditions at a gate and then propagated through the circuit. The don't care literals differ from previous attempts to handle ODCs in SAT solvers in the following ways:

- They enable these literals to be treated differently in different contexts. They can enable a clause to be ignored (effectively satisfied) early without needing to be set while deriving an implication during BCP. This is a significant difference from Velev's unobservability literals [13], which are indistinguishable from the normal literals in CNF clauses.

- We show how the don't care information is propagated and inherited during conflict driven clause learning without further referencing any circuit description. This is absent in previous approaches.

Experiments show that most circuits generate a large number of don't care literals. Efficient utilization of this information may tremendously speed up the SAT solver by both speeding up implications and guiding the decision heuristic. There is room for further improving the ideas introduced in this paper and some of the possible areas include engineering a more efficient implementation of the dynamic approach, exploiting other total orderings for the static approach, and fine-tuning the Cir-ODC parameters, some of which may be correlated to other parameters used in zChaff.

9. REFERENCES

[1] K. Bartlett, R. K. Brayton, G. Hachtel, R. M. Jacoby, C. R. Morisson, R. L. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang. Multilevel logic optimization using implicit don't cares. *IEEE Transactions on Computer-Aided Design*, pages 723–740, June, 1988.

[2] D. Brand. Verification of large synthesized designs. In *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design, Santa Clara, CA*, Novemeber, 1993.

[3] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran. In *Special Session on ATPG and Fault Simulation, International Symposium on Circuits and Systems*, June, 1985.

[4] F. Corno, M. S. Reorda, and G. Squillero. RT-Level ITC 99 benchmarks and first ATPG results. In *IEEE Design and Test of Computers*, pages 44–53, July-August, 2000.

[5] DIMACS. Satisfiability suggested format, available from `ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/doc/satformat.tex`, May, 1993.

[6] M. K. Ganai, L. Zhang, P. Ashar, A. Gupta, and S. Malik. Combining strengths of circuit-based and CNF-based algorithms for a high-performance SAT solver. In *39th Design Automation Conference*, June, 2002.

[7] A. Gupta, A. Gupta, Z. Yang, and P. Ashar. Dynamic detection and removal of inactive clauses in SAT with application in image computation. In *38th Design Automation Conference*, June, 2001.

[8] A. Kuehlmann, M. K. Ganai, and V. Paruthi. Circuit-based Boolean reasoning. In *38th Design Automation Conference*, June, 2001.

[9] F. Lu, L.-C. Wang, K.-T. T. Cheng, J. Moondanos, and Z. Hanna. A signal correlation guided ATPG solver and its applications for solving difficult industrial cases. In *40th Design Automation Conference*, June, 2003.

[10] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *39th Design Automation Conference, Las Vegas*, June, 2001.

[11] S. Safarpour, A. Veneris, R. Drechsler, and J. Lee. Managing don't cares in Boolean satisfiability. In *Design Automation and Test in Europe, Paris, France*, March, 2004.

[12] G. S. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, pages 115–125, 1968.

[13] M. N. Velev. Encoding global unobservability for efficient translation to SAT. In *The Seventh International Conference on Theory and Applications of Satisfiability Testing, Vancover, Canada*, May, 2004.