Notes on Satisfiability-Based Problem Solving

Conflict-Driven Clause Learning (CDCL)

David Mitchell
mitchell@cs.sfu.ca

February 5, 2020

This is a preliminary draft. Please do not distribute without permission. Corrections and suggestions are welcome.

We describe the conflict-driven clause learning (CDCL) algorithm for SAT, which is the basis for most high-performance SAT solvers and for many solvers for related problems.

1 Introduction

The CDCL algorithm combines, in a particular way, elements of backtracking, unit propagation, and resolution derivation.

Backtracking involves incrementally creating a partial truth assignment by choosing (guessing) assignments to particular atoms, and then retracting and reversing a guess when it has been determined that a particular partial assignment cannot be extended to a satisfying assignment. Unit propagation is an efficient method for determining that the values of some atoms are implied by others that we know. It is natural to combine these two methods. Deciding to assign P true is equivalent to adding the unit clause (P) to the clause set, and we can use unit propagation to infer other values implied by this choice. For example, if C is the clause $(\neg P \vee Q)$, and we decide to assign P true, then C in effect becomes the unit clause (Q), because the literal $\neg P$ is false. In the CDCL algorithm, guessing and unit propagation combined like this, and in addition resolution derivation is used to obtain new clauses, and these new clauses are used to retract assignments in a way that seems smarter than backtracking.

The CDCL algorithm mainly consists of alternate application of two processes. The first process incrementally constructs a truth assignment, in hopes of satisfying the clause set. It proceeds by alternately guessing a value for an unassigned literal, and then applying unit propagation to determine if the values of some further literals are implied. In the (usual) case that at some point the truth assignment being constructed makes a clause

false, the second process constructs a resolution derivation, based on the steps taken by the first process, of a new clause. This clause, in a sense, assigns blame for the failure to a particular partial truth assignment. This new clause will be added to the clause set and is also used to revise the truth assignment. The algorithm terminates when the empty clause is derived (thus establishing unsatisfiability) or a satisfying assignment is found.

Terms and Conventions In this section Γ denotes a set of m clauses, over a set S of n distinct atoms, with total length (number of literals) l. We will often write CNF formulas using a sparse notation in which $(P \vee \neg Q) \wedge (R \vee \neg S)$ is written $(P, \overline{Q}), (R, \overline{S})$.

2 Learning from Guessing

Here we describe the two main processes in CDCL. The first simply extends a given partial truth assignment, by alternately guessing a value for an unassigned atom and running unit propagation. In addition, it records the "reason" for each assignment made. This information is used by the second process, which derives a new clause.

Definition 1. An assignment stack α for clause set Γ is a sequence of literals $\alpha = l_1, l_2, \ldots l_s$ satisfying the following:

1. For each literal l_i in α, l_i or $\overline{l_i}$ appears in Γ;
2. No two literals in α are identical or are complements of each other;
3. Each literal in α has an associated "reason", which is either a clause of Γ or the special symbol **d**;
4. If the reason for literal l_i is **d**, then l_i is a decision (or guess);
5. If the reason for literal l_i is a clause C then $l_i \in C$, and every literal in C other than l_i is $\neg l_j$ for some $j < i$. That is, the assignment $l_1, \ldots l_{i-1}$ makes C effectively the unit clause (l_i).

Algorithm 1 Extend Assignment

```
1: // Assumes α is an assignment stack for Γ with no conflict.
2: // Extends α to conflict or satisfaction.
3: procedure EXTEND(Γ, α)
4:     while α ⊭ Γ and there is no C ∈ Γ with α(C) = false do
5:         L ← a literal of Γ such that α(L) is undefined
6:         α ← α, L
7:         Reason(L) ← d
8:         // Now find implications by unit propagation.
9:         while α does not make a clause false and there is an implicit unit clause do
10:            C ← a clause that is effectively unit
11:            L ← the non-false literal of C
12:            α ← α, L
13:            Reason(L) ← C
14:        end while
15:    end while
16: end procedure
```

Given an assignment stack α for Γ, the algorithm Extend, shown in Algorithm 1, extends α by alternately guessing and applying unit propagation, until either α satisfies Γ or α makes a clause of Γ false.

If an assignment stack α for Γ makes a clause of Γ false, we say there is a conflict. The algorithm Derive, shown in Algorithm 2, takes as input a clause set Γ and an assignment stack α for Γ that has a conflict. It uses the reasons for the assignments made in α to derived a sequence of clauses by resolution.

To help in understanding the Derive algorithm, we give some examples showing how one can derive new clauses based on the information kept by the Extend algorithm.

Example 1. *Consider the clauses set Γ given in Figure 1. If we run unit propagation on this clause set, and record the reason for each literal we give a value to, we find the following. First, we must set P true, with reason (P). A consequence of setting P true is that Q must be set true, with reason $(\neg Q \vee P)$. After setting Q true, we search for occurrences of $\neg Q$, which leads to setting R true with reason $(\neg Q \vee R)$ and setting $\neg S$ true with reason $(\neg Q \vee \neg S)$. Next we search for occurrences of $\neg R$. During this search (which we are assuming is from left-to-right as the clauses are written), we find that the clause $(\neg P \vee \neg R \vee S)$ has been made false, so we have a conflict and stop. We may now construct a resolution refutation of Γ based on this conflict and the reasons*

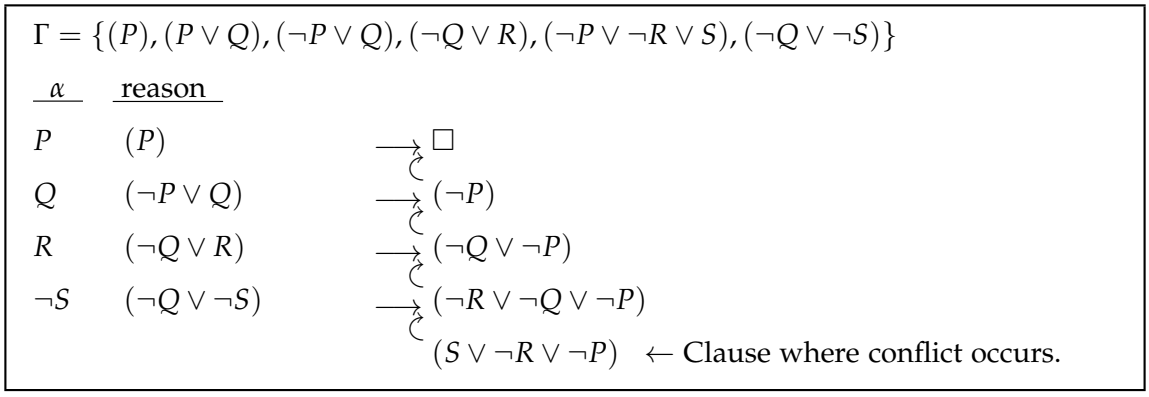

Figure 1: Conflict clause derivation example where conflict is detected based on unit propagation alone.

recorded as we built up α. The last literal of this clause to have been set false is S. The reason for $\neg S$ must be a clause in which $\neg S$ occurs, and which was effectively a unit clause before we set $\neg S$ true. Since all other literals in both of these clauses are made false by α, and they clash on S, they must resolve together. In fact, this is the case: the reason for $\neg S$ is the clause $(\neg Q \vee \neg S)$, and it resolves with $(\neg P \vee \neg R \vee S)$ to produce $(\neg R \vee \neg Q \vee \neg P)$. This clause is also made false by α, and we can apply similar reasoning to identify a clause we can resolve with it. Repeating this, we eventually derive the empty clause, giving us a resolution refutation of Γ.

Algorithm 2 Derive Asserting Clause from Conflict

```
1: // Assumes α is an assignment sequence for Γ with a conflict.
2: // Derives an asserting clause for Γ, α.
3: procedure DERIVE(Γ, α)
4:     C ← a clause of Γ such that α(C) = false
5:     while C has more than one literal L such that level(L) is the conflict level do
6:         L ← the last literal of C that was made false by α
7:         B ← reason(L)
8:         C ← the resolvant of B and C
9:     end while
10: end procedure
```

3 The CDCL Algorithm for SAT

The CDCL algorithm is shown in Algorithm 3.

Algorithm 3 CDCL

```
1: // Φ is satisfiable iff CDCL(Φ) returns "SAT".
2: procedure CDCL(Φ)
3:     Γ ← Φ   // Clause set, initialized to the input clauses.
4:     α ← ⟨⟩   // Assignment sequence, initialized to empty.
5:     while □ ∉ Γ and α ⊭ Γ do
6:         Extend α until α ⊨ Γ or there is a conflict.
7:         if there is a conflict then
8:             Derive an asserting clause C.
9:             α ← the minimum prefix of α that makes every literal in C false but one.
10:            Γ ← Γ ∪ {C}.
11:        end if
12:    end while
13:    if α ⊨ Γ then
14:        Return"SAT"
15:    else
16:        return "UNSAT"
17:    end if
18: end procedure
```

Example 2. *Figure 2 shows an example of the execution of the Extand and Derive procedures, starting from an empty truth https://www.overleaf.com/project/5ba5780b2aa6860dfa02e5eb assignment, and ending with derivation of an asserting clause that contains literals from multiple decision levels. The execution proceeds as follows. First, unit propagation adds* $P1$ *and* $P2$ *to* $alpha$*, with reasons* $(P1)$ *and* $(P2)$*. respectively. Then the search for occurrences of* $\neg P1$ *leads to adding* $P3$ *and* $P4$ *to alpha. The search for occurrences of* $\neg P2$ *does not find any essentially unit clauses, and the searches for* $\neg P3$ *and* $\neg P4$ *likewise do not, so there is no further propagation, and this completes Decision Level 0. Next,* $P5$ *is added as a decision, after which* $P6$ *and* $P7$ *are added by propagation. Next* $P8$ *is a decision, and following this* $P9$*,* $P10$ *and* $P11$ *are added to* α *by unit propagation. The final decision, at level 4, is* $P15$*. Following this decision, a conflict will be found. To understand what happens, it is important to pay attention to the order in which propagation steps occur. In the search for occurrences of* $\neg P15$*,* $P16$ *and* $P17$ *are added to* α*. The the search for occurrences of* $\neg P16$ *does not add anything. The search for occurrences of* $\neg P17$ *adds* $P18$*. The search for occurrences of* $\neg P18$ *adds* $P19$ *and* $P20$*. Next is the search for occurrences of* $\neg P19$*, which adds* $P21$*, and* $P22$*. The search for occurrences of* $\neg P20$ *adds nothing. The search for occurrences of* $\neg P21$ *adds* $P23$ *and* $P24$*. The search for occurrences of* $\neg P22$ *finds nothing. Finally, the search for occurrences of* $\neg 23$ *adds* $P25$*, and then discovers that* $(\neg P24 \vee \neg P23 \vee \neg P18)$ *has been made false, thus identifying a conflict. The Derive procedure takes this clause as the initial*

value of its variable C, and after four steps obtains the asserting clause $(\neg P18 \vee \neg P10 \vee \neg P3)$.

$\Gamma = \{(P1), (P2), (P3 \vee \neg P1), (P4 \vee \neg P1), (P6 \vee \neg P5), (P7 \vee \neg P5 \vee \neg P2), (P9 \vee \neg P8),$
$(P10 \vee \neg P8), (P11 \vee \neg P10), (P13 \vee \neg P12 \vee \neg P3), (P14 \vee \neg P13), (P16 \vee \neg P5),$
$(P17 \vee \neg P15 \vee \neg P5), (P18 \vee \neg P17), (P19 \vee \neg P18 \vee \neg P10), (P20 \vee \neg P18 \vee \neg P11),$
$(P21 \vee \neg P19), (P22 \vee \neg P19 \vee \neg P5), (P23 \vee \neg P21 \vee \neg P3), (P24 \vee \neg P21),$
$(P25 \vee \neg P23), (\neg P24 \vee \neg P23 \vee \neg P18)\}$

	α	reason
Level 0	$P1$	$(P1)$
	$P2$	$(P2)$
	$P3$	$(P3 \vee \neg P1)$
	$P4$	$(P4 \vee \neg P1)$
Level 1	$P5$	**d**
	$P6$	$(P6 \vee \neg P5)$
	$P7$	$(P7 \vee \neg P5 \vee \neg P2)$
Level 2	$P8$	**d**
	$P9$	$(P9 \vee \neg P8)$
	$P10$	$(P10 \vee \neg P8)$
	$P11$	$(P11 \vee \neg P10)$
Level 3	$P12$	**d**
	$P13$	$(P13 \vee \neg P12 \vee \neg P3)$
	$P14$	$(P14 \vee \neg P13)$
Level 4	$P15$	**d**
	$P16$	$(P16 \vee \neg P15)$
	$P17$	$(P17 \vee \neg P15 \vee \neg P5)$
	$P18$	$(P18 \vee \neg P17)$
	$P19$	$(P19 \vee \neg P18 \vee \neg P10)$
	$P20$	$(P20 \vee \neg P18 \vee \neg P11)$
	$P21$	$(P21 \vee \neg P19)$
	$P22$	$(P22 \vee \neg P19 \vee \neg P5)$
	$P23$	$(P23 \vee \neg P21 \vee \neg P3)$
	$P24$	$(P24 \vee \neg P21)$
	$P25$	$(P25 \vee \neg P23)$

Derived Asserting Clause

$(\neg P18 \vee \neg P10 \vee \neg P3)$

$(\neg P19 \vee \neg P18 \vee \neg P3)$

$(\neg P21 \vee \neg P18 \vee \neg P3)$

$(\neg P23 \vee \neg P21 \vee \neg P18)$

Conflict clause $(\neg P24 \vee \neg P23 \vee \neg P18)$

Figure 2: CDCL asserting clause derivation example with multiple decision levels.