CSE 459/598, Spring 2009

Handout 3

The Davis-Putnam-Logemann-Loveland Procedure

The *propositional satisfiability problem (SAT)* is the problem of deciding whether a given finite set of propositional formulas is satisfiable. A *SAT solver* is a software tool for solving SAT. For instance, one such tool available today is zCHAFF (`http://www.princeton.edu/∼chaff/zchaff.html`). Many existing SAT solvers, including zCHAFF, are based on the Davis-Putnam-Logemann-Loveland procedure (DPLL), invented in 1962.

Recall that a *literal* is an atom or the negation of an atom. For any atom A, the literals A, $\neg A$ are *complementary* to each other. For any literal L, the literal complementary to L will be denoted by $\overline{L}$. A *clause* is a finite set of literals. We will identify a clause with the disjunction of its elements (with $\bot$ if the clause is empty).

Any formula can be transformed into an equivalent set of clauses by the algorithm shown in Figure 1. This algorithm reduces the satisfiability problem for arbitrary sets of formulas to the satisfiability problem for sets of clauses. But CLAUSIFY(F) can be much longer than F; for instance, if F is

$$(p_1 \wedge q_1) \vee \cdots \vee (p_n \wedge q_n) \tag{1}$$

then CLAUSIFY(F) consists of 2^n clauses.

For this reason, of interest are other methods for reducing the general satisfiability problem to the clausal case. A formula F can be transformed into a "small" set of clauses Γ that is satisfiable iff F is satisfiable; moreover, given an interpretation satisfying Γ, it is easy to find an interpretation satisfying F. Unlike CLAUSIFY(F), Γ is, generally, not equivalent to F. Nevertheless, an algorithm for computing such Γ can be a useful tool for reducing the general satisfiability problem to the case of clauses.

One algorithm of this kind is given by recursive procedure CLAUSIFY* (Figure 2). Its arguments are a formula F and a set Γ of clauses, and it returns a set of clauses. To solve the satisfiability problem for a formula F_0, we invoke this procedure with $F = F_0$ and $\Gamma = \emptyset$. Generally, the clauses in CLAUSIFY$^*(F_0, \emptyset)$ contain atoms that do not occur in F_0. Any interpretation satisfying F can be extended to these "new" atoms so that CLAUSIFY$^*(F_0, \emptyset)$ will be satisfied. On the other hand, any interpretation

```
CLAUSIFY($F$)
    eliminate from $F$ all connectives other than $\neg$, $\wedge$ and $\vee$;
    distribute $\neg$ over $\wedge$ and $\vee$ until it applies to atoms only;
    distribute $\vee$ over $\wedge$ until it applies to literals only;
    return the set of conjunctive terms of the resulting formula
```

Figure 1: Clausification

```
CLAUSIFY*($F, \Gamma$)
    if $F$ is a conjunction of clauses $C_1 \wedge \cdots \wedge C_k$
        then exit with $\{C_1, \ldots, C_n\} \cup \Gamma$;
    $G :=$ a minimal non-literal subformula of $F$;
    $u :=$ a new atom;
    $F :=$ the result of replacing $G$ in $F$ by $u$;
    CLAUSIFY*($F, \Gamma \cup$ CLAUSIFY($u \leftrightarrow G$))
```

Figure 2: Clausification using new atoms

satisfying CLAUSIFY*$(F_0, \emptyset)$ becomes a model of F when restricted to the "old" atoms.

3.1^c (a) Apply CLAUSIFY* to formula (1). (b) For which values of n is the resulting set of clauses smaller than the result of applying CLAUSIFY to the same formula? (We measure the size of a set of clauses by the total number of literals in these clauses.)

A *unit* clause is a clause that consists of a single literal. If a set of clauses contains a unit clause then it can be simplified using the fact that $\{F, F \vee G\}$ is equivalent to $\{F\}$, and $\{F, \neg F \vee G\}$ is equivalent to $\{F, G\}$. A simplification step like this may create a new unit clause, and that can make further simplifications possible. This process is called *unit clause propagation*.

In the unit clause propagation procedure shown in Figure 3, Γ is a set of clauses, and U is a consistent set of literals such that, for any literal $L \in U$, neither L nor $\overline{L}$ occurs in any clause in Γ. During the execution of the procedure, Γ is simplified and U grows bigger, while the union $\Gamma \cup U$ remains equivalent to what it was originally. Upon termination, there are no unit clauses in Γ. To apply unit clause propagation to a given set Γ_0 of literals, UNIT-PROPAGATE is invoked with $\Gamma = \Gamma_0$ and $U = \emptyset$. Then, after every execution of the while loop, $\Gamma \cup U$ is equivalent to Γ_0.

```
Unit-Propagate($\Gamma, U$)
    while there is a unit clause $\{L\}$ in $\Gamma$
        $U := U \cup \{L\}$;
        for every clause $C \in \Gamma$
            if $L \in C$ then $\Gamma := \Gamma \setminus \{C\}$
            else if $\overline{L} \in C$ then $\Gamma := \Gamma \setminus \{C\} \cup \{C \setminus \{\overline{L}\}\}$
        end for
    end while
```

Figure 3: Unit clause propagation

Example 1. To apply unit clause propagation to the set

$$\{p, \neg p \vee \neg q, \neg q \vee r\} \tag{2}$$

we invoke Unit-Propagate with (2) as Γ and with the empty U. After the first execution of the body of the while loop,

$$\Gamma = \{\neg q, \neg q \vee r\} \text{ and } U = \{p\},$$

and after the second iteration

$$\Gamma = \emptyset \text{ and } U = \{p, \neg q\}.$$

This computation shows that (2) is equivalent to $\{p, \neg q\}$.

There are two cases when the process of unit clause propagation alone is sufficient for solving the satisfiability problem for Γ_0. Consider the values of Γ and U upon the termination of Unit-Propagate. *Case 1:* Γ includes the empty clause. Then Γ is unsatisfiable, and so is Γ_0. *Case 2:* $\Gamma = \emptyset$. Then Γ_0 is equivalent to U, which is a consistent set of literals; a model of Γ_0 can be easily extracted from U. For instance, the computation from Example 1 shows that any interpretation which maps p to t and q to f is a model of (2).

The Davis-Putnam-Logemann-Loveland procedure (Figure 4) is an extension of the unit clause propagation method that makes it capable of soving the satisfiability problem in full generality. For any set Γ of formulas and any formula F, the set of models of Γ is the union of the set of models of $\Gamma \cup \{F\}$ and the set of models of $\Gamma \cup \{\neg F\}$. The DPLL procedure uses this fact to apply unit clause propagation even when Γ does not contain unit clauses.

```
DPLL($\Gamma, U$)
    UNIT-PROPAGATE($\Gamma, U$);
    **if** $\emptyset \in \Gamma$ **then** return;
    **if** $\Gamma = \emptyset$ **then** exit with a model of $U$;
    $L :=$ a literal such that $L$ or $\overline{L}$ occurs in $\Gamma$;
    DPLL($\Gamma \cup \{L\}, U$);
    DPLL($\Gamma \cup \{\overline{L}\}, U$)
```

Figure 4: Davis-Putnam-Logemann-Loveland procedure

In this recursive procedure, Γ is again a set of clauses, and U is a consistent set of literals such that, for any literal $L \in U$, neither L nor $\overline{L}$ occurs in any clause in Γ. The return in the first if statement of DPLL indicates that $\Gamma \cup U$ is unsatisfiable. The exit in the second if statement produces a model of $\Gamma \cup U$. Initially, DPLL is invoked with $\Gamma = \Gamma_0$ and $U = \emptyset$.

Example 2. Consider the application of the DPLL procedure to the set

$$\{\neg p \vee q, \neg p \vee r, q \vee r, \neg q \vee \neg r\}. \tag{3}$$

First DPLL is called with (3) as Γ and the empty U (Call 1). After the call to UNIT-PROPAGATE, the values of Γ and U remain the same. Assume that the literal selected as L is p. Now DPLL is called recursively with

$$\{\neg p \vee q, \neg p \vee r, q \vee r, \neg q \vee \neg r, p\}$$

as Γ and $U = \emptyset$ (Call 2). After the call to UNIT-PROPAGATE, $\emptyset \in \Gamma$. Next DPLL is called with

$$\{\neg p \vee q, \neg p \vee r, q \vee r, \neg q \vee \neg r, \neg p\}$$

as Γ and $U = \emptyset$ (Call 3). After the call to UNIT-PROPAGATE, Γ is

$$\{q \vee r, \neg q \vee \neg r\}$$

and $U = \{\neg p\}$. Assume that the literal selected as L is q. Then DPLL is called with

$$\{q \vee r, \neg q \vee \neg r, q\}$$

as Γ and $U = \{\neg p\}$ (Call 4). After the call to UNIT-PROPAGATE, $\Gamma = \emptyset$ and $U = \{\neg p, q, \neg r\}$. The computation produces the model of (3) that assigns the value t to q, and the value f to both p and r.

3.2^c How would the computation described in Example 2 be affected (a) by selecting $\neg p$ as L in Call 1? (b) by selecting $\neg q$ as L in Call 3?