Instructor: Edmund M. Clarke Teaching Assistant: Michael Carl Tschantz

Assignment 2

1 DNF

Put $(x \to y) \wedge (z \leftrightarrow x)$ into disjunctive normal form.

2 CNF

Let f be the DNF formula $(u \wedge v) \vee (w \wedge x)$.

(a) Find a formula in conjunctive normal form that is logically equivalent to f.

(b) Find an equi-satisfiable formula in CNF to f by introducing a new variable a for $u \wedge v$ and a new variable b for $w \wedge x$.

3 Universal Sets of Connectives

We call a set of connectives *universal* if for any logical formula, there exists an equivalent formula using only connectives in that set. For example, $\{\wedge, \vee, \neg\}$ is a universal set.

Let $\otimes$ be the NAND connective with the following truth table

x	y	$x \otimes y$
F	F	T
F	T	T
T	F	T
T	T	F

(a) Produce a formula equivalent to $\neg x$ using only the connective $\otimes$ and variable x.

(b) Produce a formula equivalent to $x \wedge y$ using only the connective $\otimes$ and the variables x and y.

(c) Produce a formula equivalent to $x \vee y$ using only the connective $\otimes$ and variables x and y.

(d) Briefly explain how this shows that $\{\otimes\}$ is a universal set of connectives.

4 Davis-Putnam

In this problem you will run the Davis-Putnam algorithm by hand showing the value of the input variable S and the return value for each call to `Satisfiable`.

Background. The Davis-Putnam algorithm is provided in the lecture "Propositional Logic" (page 22) as the function `Satisfiable`. The function `Satisfiable` has one input, the clause list S. Since the algorithm operates on a formula in CNF, the clauses of S are implicitly conjoined, and each clause of S represents a disjunction literals. For example, the formula $(\neg x \vee y) \wedge (x \vee \neg y) \wedge (y \vee z) \wedge (x \vee \neg z)$ is represented as the list of lists $[[\neg x, y], \quad [x, \neg y], \quad [y, z], \quad [x, \neg z]]$.

Note that line 12 of the algorithm

`choose a literal` L `occurring in` S

introduces nondeterminism into the system since it does not state which literal L to choose. To remove this nondeterminism, presume that the algorithm always selects the first literal of the first clause in S.

Treat $S \cup \{L\}$ as appending the list S to the list $[[L]]$. For example, $[[x, \neg y], \quad [x, z]] \cup \{z\}$ is $[[x, \neg y], \quad [x, z], \quad [z]]$.

What You Must Do. Run the function `Satisfiable` by hand when called with value of $[[\neg x, y], \quad [x, \neg y], \quad [y, z], \quad [x, \neg z]]$ for S. Since `Satisfiable` is a recursive function (see lines 13 and 14), it will call itself. To prove that you ran the function by hand, show the value of the input S at the beginning of each of these recursive calls to `Satisfiable`. Also, note the return value of each call.

Examples. For example, if the problem asked you to run `Satisfiable` with the initial value of $[[\neg x, y], \quad [y, \neg z], \quad [\neg y, x]]$ for S instead of $[[\neg x, y], \quad [x, \neg y], \quad [y, z], \quad [x, \neg z]]$, a valid solution to this problem would be (with optional comments explaining the solution in parentheses):

> (`Satisfiable` is initially called with $S = [[\neg x, y], \quad [y, \neg z], \quad [\neg y, x]]$. It selects $\neg x$ for L on line 12 for the first recursive call on line 13.)
>
> On the first and only recursive call of `Satisfiable`, $S = [[\neg x, y], \quad [y, \neg z], \quad [\neg y, x], \quad [\neg x]]$. ($[\neg x]$ is a unit clause so `Satisfiable` deletes every clause containing $\neg x$, and x is deleted from every remaining clause. This yields $[[y, \neg z], \quad [\neg y]]$. Now $[\neg y]$ is a unit clause so every clause containing $\neg y$ is deleted, and y is deleted from every remaining clause. This yields $[[\neg z]]$. Since $[\neg z]$ is a unit clause it is deleted leaving []. Since S is now empty,) true is returned by this recursive call.

Another example: if the problem instead asked you to run `Satisfiable` with the initial value of $[[\neg x, y, z], \quad [x, \neg y, \neg z], \quad [y, z], \quad [x, y, \neg z]]$ for S, a valid solution would be:

> (`Satisfiable` is initially called with $S = [[\neg x, y, z], \quad [x, \neg y, \neg z], \quad [y, z], \quad [x, y, \neg z]]$. It selects $\neg x$ as L for the first recursive call.)
>
> On the first recursive call, $S = [[\neg x, y, z], \quad [x, \neg y, \neg z], \quad [y, z], \quad [x, y, \neg z], \quad [\neg x]]$. ($[\neg x]$ is a unit clause so every clause containing $\neg x$ is deleted, and x is deleted from every remaining clause. This yields $[[\neg y, \neg z], \quad [y, z], \quad [y, \neg z]]$. $\neg y$ is selected as L for the second recursive call.) True is returned by this recursive call. (The reason being that true is returned the third recursive call, which is described below.)
>
> On the second recursive call, $S = [[\neg y, \neg z], \quad [y, z], \quad [y, \neg z], \quad [\neg y]]$. ($[\neg y]$ is a unit clause so this yields $[[z], \quad [\neg z]]$ after the removals. z is a unit clause yielding $[[]]$. Thus, S contains the null clause.) False is returned by this recursive call. (y is now tried by the first recursive call since $[\neg y]$ lead to false.)
>
> On the third and final recursive call, $S = [[\neg y, \neg z], \quad [y, z], \quad [y, \neg z], \quad [y]]$. ($[y]$ is a unit clause yielding $[[\neg z]]$. Since $[[\neg z]]$ is a unit clause, it is removed leaving [].) True is returned by this call.