1 Answers for Assignment 3 [1]

1.1

The truth table for $f = \neg(x + \neg y \cdot (\neg x + z))$ is given by

x	y	z	f
F	F	F	F
F	F	T	F
F	T	F	T
F	T	T	T
T	F	F	F
T	F	T	F
T	T	F	F
T	T	T	F

1.2

In the truth table we see that f is true for the expression $\overline{x}y\overline{z} + \overline{x}yz$. However, this expression is not in CNF, but in DNF.

Rather than convert this expression from DNF to CNF, we make use of two facts

- $f \equiv \overline{\overline{f}}$
- By DeMorgan's law, the negation of a formula in DNF form yields a formula in CNF form.

$\overline{f}$ is true for the expression $\overline{xyz} + \overline{xy}z + x\overline{yz} + x\overline{y}z + xy\overline{z} + xyz$
By the first fact, we have $f \equiv \overline{\overline{f}} \equiv \overline{\overline{xyz} + \overline{xy}z + x\overline{yz} + x\overline{y}z + xy\overline{z} + xyz}$.
Applying DeMorgan's Law to get a CNF form, we have
$f \equiv (x + y + z)(x + y + \overline{z})(\overline{x} + y + z)(\overline{x} + y + \overline{z})(\overline{x} + \overline{y} + z)(\overline{x} + \overline{y} + \overline{z})$

1.3

The DPLL algorithm tries two steps before making recursive calls.

- check for unit clauses (UC)
- check for pure variables (PC)

$DPLL(f)$:
$UC(f)$? fail
$PC(f)$? fail
DPLL branches using assignments of T and F to x.
$x \leftarrow T$

$$f_T = (y + z)(y + \overline{z})(\overline{y} + z)(\overline{y} + \overline{z})$$

[1]Found an error? Please contact Kedar Swadi(kswadi@cs.princeton.edu)

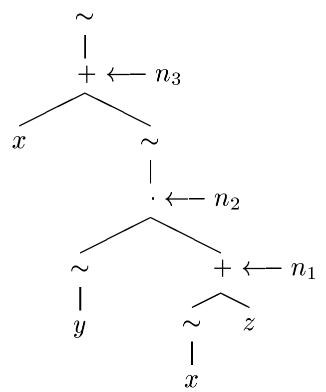
$DPLL(f_T)$:
$UC(f_T)$? fail
$PC(f_T)$? fail
Branch on y
$y \leftarrow T$
$DPLL(f_{TT}) = (z)(\overline{z})$
    $UC(f_{TT})$? success
    Performing unit propogation on f_{TT} gives $f'_{TT} = ()$
    Since we have an empty clause, we fail.
$y \leftarrow F$
$DPLL(f_{TF}) = (\overline{z})(z)$
    $UC(f_{TF})$? success
    Performing unit propogation on f_{TF} gives $f'_{TF} = ()$.
    Since we have an empty clause, we fail.

$x \leftarrow F$
$f_F = (y + z)(y + \overline{z})$
$DPLL(f_F)$:
$UC(f_F)$? fail
$PC(f_F)$? Succeed for $y = T$

Hence we have $x = F, y = T, z = \{T, F\}$ as a solution.

2

The central idea in this problem is to "invent" new variables to create a $3-CNF$ formula f' from the formula f such that f has a solution iff f' has a solution. It is useful to look at an expression as a tree.



In this tree, we associate new variables n_1, n_2, n_3 with each of the binary branch nodes. This helps us to associate a third variable with the two variables involved in the binary operation. Consider the subexpression $(\neg x + z)$.

We could write f informally as $f = \neg(x + \neg y \cdot n_1)$ where n_1 *has the same value* as $(\neg x + z)$. This is formally written using the "equivalence" sign as $n_1 \equiv (\neg x + z)$.

$a \equiv b$ says that both a and b are true or both are false. This is the same as saying $(a \cdot b) + (\overline{a} \cdot \overline{b})$. Using the double negation trick in section 1, we can write it as $(a + \overline{b})(\overline{a} + b)$ which gives us a CNF form.

Performing this operation repeatedly, we have
$f \equiv \neg(n_3)$ where

- $n_3 \equiv (x + \neg(n_2))$,
- $n_2 \equiv (\neg(y) \cdot n_1)$, and
- $n_1 \equiv (\neg x + z)$

Our original formula is satisfied if $\neg(n_3)$ holds and the three formulae above also hold. Using techniques described in the earlier question, we get 3-CNF formulae for each of the three equivalences above. Assume these formulae are f_1, f_2 and f_3. (In your answers you are expected to show the derivations of these formulae).

Therefore the answer is $\neg(n_3) \cdot f_1 \cdot f_2 \cdot f_3$.

3

Encoding the n-Queens problem as as SAT problem consists of encoding constraints on the cells of the board.

We make n^2 boolean variables $q_{i,j}$ $(1 \leq i \leq n, 1 \leq j \leq n)$, one for each cell (i, j) of the board.

$$q_{i,j} = \text{True if cell } (i,j) \text{ has a queen}$$
$$q_{i,j} = \text{False otherwise}$$

A legal n-Queens board is one where no two queens attack each other. This condition may be broken down into three conditions

- No queen attacks another queen in a row.i.e. no row has more than one queen. However, by the pigeon hole principle, we see that each row must have at least one queen. Hence we have the condition that each row has exactly one queen. Consider the row k. We encode the previous statement as

$$\begin{array}{ll} (q_{k1} \cdot \overline{q_{k2}} \cdot \overline{q_{k3}} \ldots \overline{q_{kn}}) \vee & \text{(queen in col 1 and no other col)} \\ (\overline{q_{k1}} \cdot q_{k2} \cdot \overline{q_{k3}} \ldots \overline{q_{kn}}) \vee & \text{(queen in col 2 and no other col)} \\ \vdots & \\ (\overline{q_{k1}} \cdot \overline{q_{k2}} \cdot \overline{q_{k3}} \ldots q_{kn}) & \text{(queen in col } n \text{ and no other col)} \end{array}$$

  This can be written more formally as $\bigvee_{1 \leq j \leq n}(q_{kj} \cdot \bigwedge_{1 \leq i \leq n, i \neq j} \overline{q_{ki}})$. For all the rows, we make a conjunction of these conditions. This may be written as $\bigwedge_{1 \leq r \leq n}(\bigvee_{1 \leq j \leq n}(q_{rj} \cdot \bigwedge_{1 \leq i \leq n, i \neq j} \overline{q_{ri}}))$.

- No queen attacks another queen in a column : The encoding is analogous to the one for the rows.

- No queen attacks another queen in a diagonal: The diagonal case is a little tricky. We first note the relation between two cells on a diagonal. We see that the "gradient" for diagonals is either 1 or -1. A simpler way to put it is that if two cells (i_1, j_1) and (i_2, j_2) are on a "left to right" diagonal, then $i_1 - i_2 = j_1 - j_2$. For a "right to left" diagonal, we have $i_2 - i_1 = j_1 - j_2$. Unlike rows or columns, it is not clear that we have *exactly* one queen per diagonal. However, we may have *at most* one queen per diagonal. We may encode this condition by saying that any two cells on a diagonal should not both have queens. For cells (i_1, j_1) and (i_2, j_2) on a diagonal, we write this as $(\overline{q_{i_1 j_1}} + \overline{q_{i_2 j_2}})$ (either $q_{i_1 j_1}$ has no queen or $q_{i_2 j_2}$ has no queen). This may be written for the "right to left" diagonals as $\bigwedge_{(1 \leq i_1, j_1, i_2, j_2 \leq n, i_1 - i_2 = j_1 - j_2)} (\overline{q_{i_1 j_1}} + \overline{q_{i_2 j_2}})$. For the other diagonals, we have $\bigwedge_{(1 \leq i_1, j_1, i_2, j_2 \leq n, i_2 - i_1 = j_1 - j_2)} (\overline{q_{i_1 j_1}} + \overline{q_{i_2 j_2}})$.

A conjunction of these conditions encode the $n-$Queens problem in SAT.