

# Tic-Tac-Toe with SAT

William John Holden

September 22, 2019

## 1 Introduction

In this exercise we will construct a series of boolean satisfiability clauses to find a winning move for the game Tic-Tac-Toe. We treat the SAT solver as a “black box” and will not worry about how it works. A SAT solver accepts a *formula of clauses* containing boolean *variables* in *conjunctive normal form* (CNF). An example of a formula in CNF is

$$(A \vee \neg B)(C)(C \vee D \vee E)(B \vee \neg C). \quad (1)$$

This formula is *satisfiable* with the literals  $A = T$ ,  $B = T$ ,  $C = T$ ,  $D = F$ , and  $E = F$ . This is not the only satisfying assignment.  $D$  and  $F$  may each take the values  $T$  or  $F$ , but we are constrained in  $C = T$ . If  $C = F$  then the clause  $(C)$  cannot be satisfied, and any unsatisfied clause falsifies the entire formula.

Some formulas cannot be satisfied under any assignment of literals to variables. An obvious example is  $(A)(\neg A)$ . A less obvious example is

$$(A \vee \neg B)(B \vee \neg C)(C)(\neg A \vee \neg B \vee \neg C). \quad (2)$$

In this case, the  $(C)$  clause implies  $C = T$ , which means  $B = T$ , which then implies  $A = T$ , but now the final clause is  $(F \vee F \vee F)$  which falsifies the formula.

A computer program for finding a set of satisfying literals, or proving that no such assignment exists, is called a *SAT solver*. Though the boolean satisfiability problem is NP-complete, state of the art SAT solvers are very efficient in practice. Martin Horenovsky has a very nice article at <https://codingnest.com/modern-sat-solvers-fast-neat-underused-part-1-of-n/> on this subject. I originally learned of the satisfiability problem and SAT solvers attending a course on NP-Complete Problems from Alexander Kulikov (<https://www.edx.org/course/np-complete-problems-uc-san-diegox-algs203x>).

Others have successfully solved Sudoku puzzles using SAT solvers. In this exercise, we will do the same for tic-tac-toe. Of course, tic-tac-toe is much less computationally demanding than Sudoku. Tic-tac-toe has a smaller board ( $3 \times 3$  instead of  $9 \times 9$ ) and it has fewer constraints. If we consider the size of the tic-tac-toe puzzle to be  $n$  for dimensions  $\sqrt{n}$  rows  $\times$   $\sqrt{n}$  columns  $= x \times y$  then we can iterate over every row and column to find a winning move in

$$O(x \times y) + O(y \times x) + O(2 \text{ diagonals} \times \sqrt{n}) = \quad (3)$$

$$O(xy) + O(xy) + O(2\sqrt{n}) = O(n). \quad (4)$$

Using an exponential time tool for a linear time problem may seem crazy, but the goal here is to learn to *reduce* problems to SAT.

## 2 Reduction to SAT

### 2.1 Variables

We need to express the positions of our tic-tac-toe board as boolean variables. This is pretty easy. We assign a variable letter to each position in reading order:

$a$	$b$	$c$
$d$	$e$	$f$
$g$	$h$	$i$

There are actually *three* possible states for each position: X, O, and unassigned. However, in this problem we do not need to concern ourselves with the third state. We will consider a variable to be “true” if it is marked with X and false otherwise.

### 2.2 Constraint 1: Winning Combinations

There are eight total winning combinations in tic-tac-toe: satisfy any of the three rows, any of the columns, or either of the diagonals. I should note that the word *or* in the previous sentence is an inclusive or; winning combinations allow for both a row and a column, both a row and a diagonal, and both a column and a diagonal. For example, the assignment

$o$	$o$	$x$
$o$	$o$	$x$
$x$	$x$	

can be won by setting  $i = x$ . We can express the eight winning combinations as

$$(a \wedge b \wedge c) \vee (d \wedge e \wedge f) \vee (g \wedge h \wedge i) \vee (a \wedge d \wedge g) \vee (b \wedge e \wedge h) \vee (c \wedge f \wedge i) \vee (a \wedge e \wedge i) \vee (c \wedge e \wedge g). \quad (5)$$

This long formula is in *disjunctive normal form* (DNF) and not the CNF we want.