A Lower Bound of 2^n Conditional Jumps for Boolean Satisfiability on A Random Access Machine

Samuel C. Hsieh
Computer Science Department, Ball State University

July 1, 2014

Abstract

We establish a lower bound of 2^n conditional jumps for deciding the satisfiability of the conjunction of any two Boolean formulas from a set called a *full representation* of Boolean functions of n variables - a set containing a Boolean formula to represent each Boolean function of n variables. The contradiction proof first assumes that there exists a RAM program that correctly decides the satisfiability of the conjunction of any two Boolean formulas from such a set by following an execution path that includes fewer than 2^n conditional jumps. By using multiple runs of this program, with one run for each Boolean function of n variables, the proof derives a contradiction by showing that this program is unable to correctly decide the satisfiability of the conjunction of at least one pair of Boolean formulas from a full representation of n-variable Boolean functions if the program executes fewer than 2^n conditional jumps. This lower bound of 2^n conditional jumps holds for any full representation of Boolean functions of n variables, even if a full representation consists solely of minimized Boolean formulas derived by a Boolean minimization method. We discuss why the lower bound fails to hold for satisfiability of certain restricted formulas, such as 2CNF satisfiability, XOR-SAT, and HORN-SAT. We also relate the lower bound to 3CNF satisfiability.

1 Introduction

The problem of deciding whether a Boolean formula is satisfiable is commonly known as the Boolean satisfiability problem. It was the first problem shown to be NP-complete [1]. We establish a lower bound of 2^n conditional jumps for deciding the satisfiability of the conjunction of any two Boolean formulas from a set called a *full representation* of Boolean functions of n variables - a set containing a Boolean formula to represent each Boolean function of n variables. The contradiction proof first assumes that there exists a RAM program [5] that correctly decides the satisfiability of the conjunction of any two Boolean formulas from such a set by following an execution path that includes fewer than 2^n conditional jumps. By using multiple runs of this RAM program, with one run for each Boolean function of n variables,

the proof derives a contradiction by showing that this RAM program is unable to correctly decide the satisfiability of the conjunction of at least one pair of Boolean formulas from a full representation of n-variable Boolean functions if the RAM program executes fewer than 2^n conditional jumps.

We briefly summarize the remaining sections of this paper. The next section provides a brief overview of Boolean formulas and the random access machine model due to Machtey and Young [5]. As there are variations in the nomenclatures used in the literature, an overview of the related concepts and terminology as used in this paper seems appropriate. Section 3 introduces concepts related to executing multiple runs of a RAM program and proves a few related lemmas. Section 4 proves the lower bound of 2^n conditional jumps and shows that the lower bound applies to CNF satisfiability and, by duality, to DNF falsifiability. Section 5 first discusses why the lower bound fails to hold for satisfiability of certain restricted formulas such as 2CNF satisfiability, XOR-SAT and HORN-SAT, and the section then relates the lower bound to 3CNF satisfiability. Section 6 discusses some lower bounds on two other abstract models of computation.

2 Boolean Formulas and a RAM

Boolean formulas are widely known, e.g., [2,7]. As there are variations in the nomenclatures, we summarize the related concepts and terminology as used here.

2.1 Boolean Formulas

The set $B = \{true, false\}$ denotes the set of *Boolean values*. A Boolean *variable* has either $true$ or $false$ as its value. A function $f : B^n \to B$ is a Boolean function of n variables. The expression $B^n \to B$ denotes the set of Boolean functions of n variables.

We may use a Boolean *formula* to define or represent a Boolean function. A Boolean formula is composed of Boolean values, Boolean variables, and the Boolean operators $\wedge$ (for conjunction, i.e., AND), $\vee$ (for disjunction, i.e., OR) and $\overline{overbar}$ (for negation, i.e., NOT). A Boolean function can be represented by many different Boolean formulas.

A *literal* is a Boolean variable or a logically negated variable. A Boolean formula in DNF (*Disjunctive Normal Form*), or a DNF formula, is an OR of DNF clauses, and a DNF clause is an AND of literals. For example, $(\overline{x_1} \wedge x_2) \vee (x_1 \wedge \overline{x_2}) \vee (x_1 \wedge x_3)$ is a DNF formula. A DNF formula is a kDNF formula if each clause has k literals. The example just given is a 2DNF formula. A Boolean formula in CNF (*Conjunctive Normal Form*), or a CNF formula, is an AND of CNF clauses, and a CNF clause is an OR of literals. For example, $(x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (x_2 \vee x_3)$ is a CNF formula. A CNF formula is a kCNF formula if each clause has k literals. The example just given is a 2CNF formula.

An *assignment* to a set of Boolean variables assigns a Boolean value to each variable in the set. An assignment can be used to evaluate a Boolean formula or a Boolean function. If an assignment makes a formula or a function $true$, the assignment is said to *satisfy* the formula or the function and is called a *satisfying* assignment; similarly, if an assignment

makes a formula or a function $false$, the assignment is said to *falsify* the formula or the function and is called a *falsifying* assignment.

A Boolean formula is *satisfiable* if it has a satisfying assignment; otherwise, the formula is *unsatisfiable*. A Boolean formula is *falsifiable* if it has a falsifying assignment; otherwise, the formula is *unfalsifiable*.

2.2 A Random Access Machine

Henceforth the acronym *RAM* refers to the random access machine due to Machetey and Young [5]. It is a simple model of computation. The RAM is associated with an alphabet and deals with strings composed from the alphabet. In this paper, the alphabet of the RAM will be denoted by $\Sigma = \{a_1, a_2, \cdots, a_k\}$, where $k > 1$, and the set of strings composed from Σ will be denoted by Σ^*. The RAM has access to an unlimited number of registers named R_1, R_2, $\cdots$. Each register can store a string from Σ^*.

A RAM program is a finite sequence of RAM instructions. Each instruction in a RAM program can be optionally labeled by a "line name" [5], henceforth referred to as an *instruction label* or simply as a *label*. The instructions in a RAM program are executed sequentially, unless a jump is performed. A RAM instruction can be of one of the following four types. [1]

- The instruction **add** uses two operands: a register R_i and a symbol $a_j \in \Sigma$. The instruction appends the string stored in R_i with the symbol a_j. As a result, a_j becomes the new rightmost symbol of the string in R_i.

- The instruction **delete** uses one operand: a register R_i. The instruction deletes the leftmost symbol (if any) from the string stored in R_i.

- The instruction **conditional jump** uses three operands: a register R_i, a symbol $a_j \in \Sigma$, and a label *dest*. If the first (i.e., leftmost) symbol of the string in R_i is a_j, then jump to the nearest instruction above or below (as specified in the instruction) that is labeled *dest*.

- The instruction **continue** is a no-op instruction, but when used as the final instruction of a program, the instruction halts the program. Henceforth, a final continue instruction will be referred to as the **halt** instruction.

Executing a conditional jump instruction makes a binary decision: if the first symbol of the string in R_i is a_j then a jump takes place; otherwise execution continues sequentially. Program execution continues sequentially after executing an add, delete, or non-halting continue instruction. A *RAM program* is defined to be a finite sequence of RAM instructions such that a) the destination of any jump (i.e., the label to jump to) exists, and b) the final instruction is a continue instruction, which serves to halt program execution.

[1] Only a subset of RAM instructions is given here. Every RAM program using the full set of RAM instructions can be converted into an equivalent program using only the instructions in this subset.

Any input to a RAM program is to be provided as the initial contents of a finite number of registers R_1, R_2, $\cdots$, R_i, and the other registers are initially empty. When a program terminates, the output from the program is to be found in the register R_1.

For the Boolean satisfiability problem, an answer is either yes or no. We will say that a RAM program **accepts** its input if the program halts with the answer yes in R_1 and that a program **rejects** its input if the program halts with the answer no in R_1.

3 Multiple Runs of a RAM Program

First, we will define a few related terms.

Definition 1. An execution of a RAM program P with ω_1,ω_2, $\cdots$, $\omega_i \in \Sigma^*$ as the initial contents of the registers R_1, R_2, $\cdots$, R_i, respectively, is called a *run* of P and is denoted by P(ω_1, ω_2, $\cdots$, ω_i).

A run of a RAM program solves an **instance** of the general problem that the program is intended to solve. For example, if a program P solves the Boolean satisfiability problem, then a run of P decides whether a specific Boolean formula is satisfiable.

Definition 2. An *execution path*, or simply a *path*, is a sequence of instructions that a RAM program may execute, beginning with the initial instruction of the program. A path is either *terminated* or *open*: a terminated path ends with the halt instruction, and an open path ends with an instruction other than the halt instruction. A program that serially executes the entire sequence of instructions of a path is said to *follow* the path. The first instruction that a program executes after following an open path U or after executing an instruction i is said to *immediately succeed* the path U or the instruction i. An instruction that immediately succeeds a path U or an instruction i is called an *immediate successor instruction* of U or of i.

As mentioned before, executing a conditional jump instruction involves a two-way decision: if the first symbol of the string in a certain register is a certain symbol, then a jump takes place else execution continues sequentially. Hence, a conditional jump instruction may have two alternative immediate successor instructions. On the other hand, due to sequential execution, there is one **fixed** immediate successor instruction for any add, delete, or non-halting continue instruction in a RAM program. A terminated path has no successor instruction.

Lemma 1. For any RAM program and for any integer $m \geq 0$, the sum of the following two numbers is **no more than** 2^m.

a) the number of distinct open paths with each containing $m + 1$ conditional jump instructions and ending with a conditional jump instruction, and

b) the number of distinct terminated paths with each containing m or fewer conditional jump instructions.

Proof. We define a term to be used later in the proof: a *line* is a sequence of instructions that a program may execute and that consists of 0 or more add, delete or non-halting continue instructions followed either by a conditional jump instruction or by the halt instruction. In other words, a line is a piece of straight line code that ends with a conditional jump or the halt instruction. Since the instructions add, delete and non-hating continue are executed sequentially, if the first instruction of a line is executed, all instructions of the line will be executed serially. Every instruction i in any RAM program begins no more than one line. This is because if i is the halt instruction or a conditional jump, then i alone is a line; otherwise, i is an add, delete or non-halting continue instruction, and in any line that begins with i, due to sequential execution of the instructions add, delete and non-halting continue, there is only one possible sequence of execution from i and no alternative is possible until the end of the line, which is either a conditional jump or the halt instruction. Hence, an instruction cannot begin more than one line.

We will prove the lemma by induction. The initial instruction of any RAM program begins no more than one line, which ends with either the halt instruction or a conditional jump. In the former case, the line is one ($= 2^0$) terminated path containing no conditional jump; in the latter case, the line is one open path containing one conditional jump instruction. Hence the lemma holds for $m = 0$. Suppose that there are t distinct terminated paths with each containing j or fewer conditional jump instructions, that there are p open paths with each containing $j + 1$ conditional jump instructions and ending with a conditional jump instruction, and that $t + p \leq 2^j$ (inductive hypothesis). Since each of the p open paths ends with a conditional jump instruction, each such open path has no more than two alternative immediate successor instructions, each of which begins no more than one line, which either ends with the halt instruction or with a conditional jump instruction. Hence, by appending each of the p open paths with each of the lines that the open path's alternative immediate successors begin, we can form no more than $2p$ new distinct paths that end either with the halt instruction or with a conditional jump. Of the $2p$ new paths, those that end with a conditional jump instruction remain open with each containing $j + 2$ conditional jump instructions (including the new ending conditional jump instruction), and those that end with the halt instruction become terminated, with each containing $j + 1$ conditional jump instructions (as no new conditional jump instruction is added). No new paths can be formed from the t terminated paths each of which contains j or fewer conditional jump instructions, since terminated paths have no successor instruction. Hence the sum of the number of distinct open paths with each containing $j + 2$ conditional jump instructions and ending with a conditional jump, and the number of terminated paths with each containing $j + 1$ or fewer conditional jump instructions, is no more than $t + 2p$, which is no more than 2^{j+1} since by the inductive hypothesis $t + p \leq 2^j$. **Q.E.D.**

Lemma 2. For any strings $first_1$, $second_1$, $first_2$, $second_2 \in \Sigma^*$ and for any RAM program P, if P($first_1$, $second_1$) and P($first_2$, $second_2$) follow a terminated path U, then P($first_1$, $second_2$) must follow the same terminated path U.

Proof. We will prove the lemma by contradiction. Let the path U, which the two runs P($first_1$, $second_1$) and P($first_2$, $second_2$) follow, be the sequence of instructions U_1 U_2

$\cdots U_u$, let the run P($first_1$, $second_2$) follow the path Q, and let Q be the sequence of instructions $Q_1\ Q_2 \cdots Q_q$. Assume that Q is different from U. The rest of the proof will derive a contradiction to this assumption.

Since Q is different from U, there is at least one integer i such that the instruction Q_i is different from the instruction U_i. Of such integers, there must a least one. Let m be the least such integer. Since m is the smallest integer such that Q_m is different from U_m, the path $U_1 \cdots U_{m-1}$ is identical to the path $Q_1 \cdots Q_{m-1}$. Since, by definition, both paths begin with the initial instruction of the program P, $m \geq 2$. Let T be the point in U and in Q between the instruction U_{m-2} and the instruction U_{m-1} (i.e., between Q_{m-2} and Q_{m-1}). Let the three runs execute to the point T, where each run has completed the common sequence of instructions $U_1 \cdots U_{m-2}$ (i.e., $Q_1 \cdots Q_{m-2}$) but has not executed the instruction U_{m-1} (i.e., Q_{m-1}) yet. Since U_m and Q_m are different instructions, the instruction U_{m-1} (i.e., Q_{m-1}) has two different immediate successor instructions and therefore must be a conditional jump. Let the operands of the conditional jump instruction U_{m-1} be the register R_x, some symbol a_y, and some label: the conditional jump instruction U_{m-1} selects its immediate successor by checking to see whether the first symbol of the string in R_x is a_y. The register R_x can be either R_1, R_2 or some other register. In each case, U_m can be shown to be the same instruction as Q_m, as detailed below.

A) Suppose that R_x is R_1. The two runs P($first_1$, $second_1$) and P($first_1$, $second_2$) have the same string $first_1$ as the initial content of their respective R_1. As they follow the common path $U_1 \cdots U_{m-2}$ to the point T, the two runs perform an identical sequence of operations on their registers, including R_1. Hence, at the point T, the two runs will have the same string stored in their respective R_1. This will cause the conditional jump instruction U_{m-1} (i.e., Q_{m-1}) to select the same immediate successor instruction for the two runs. That is, U_m is the same instruction as Q_m.

B) Suppose that R_x is R_2. The two runs P($first_2$, $second_2$) and P($first_1$, $second_2$) have the same string $first_2$ as the initial content of their respective R_2. As they follow the common path $U_1 \cdots U_{m-2}$ to the point T, the two runs perform an identical sequence of operations on their registers, including R_2. Hence, at the point T, the two runs have the same string stored in their respective R_2. This will cause the conditional jump instruction U_{m-1} (i.e., Q_{m-1}) to select the same immediate successor instruction for the two runs. That is, U_m is the same instruction as Q_m.

C) Suppose that R_x is neither $R1$ nor R_2. Since R_x is not used to hold any input string, R_x is initially empty for all three runs. That is, all three runs have identical initial content (the null string) for their respective R_x. As the three runs follow the common path $U_1 \cdots U_{m-2}$ to the point T, the three runs perform an identical sequence of operations on their registers, including R_x. Hence, at the point T, all three runs have the same string strored in their respective R_x. This will cause the conditional jump instruction U_{m-1} (i.e., Q_{m-1}) to select the same immediate successor instruction for all three runs. That is, U_m is the same instruction as Q_m.

Thus, there does not exist an integer m such that Q_m is different from U_m. In other words, the path U and the path Q are identical. **Q.E.D.**

4 A Lower Bound for Satisfiability

We now establish a worst-case lower bound on the number of conditional jump instructions required in order for a RAM program to decide Boolean satisfiability. Obviously, the alphabet of the RAM is assumed to be appropriate for the Boolean satisfiability problem.

Definition 3. Let $x_1, x_2 \cdots x_n$ be the variables of which the members of the set $B^n \to B$ are functions. A Boolean formula that *represents* or *defines* a Boolean function $f : B^n \to B$ is a formula ϕ_f of the variables $x_1, x_2 \cdots x_n$ such that, for every assignment to the variables $x_1, x_2 \cdots x_n$, the formula ϕ_f evaluates to the same value as what the function f evaluates to.

A Boolean formula that represents a function f will be denoted by ϕ_f here. However, the symbol ϕ without a subscript, or with a numerical subscript, such as ϕ_3, will denote a Boolean formula without indicating the specific function that it represents.

Definition 4. Let $x_1, x_2 \cdots x_n$ be the variables of which the members of the set $B^n \to B$ are functions. A *full representation* of the set $B^n \to B$ is a set E of Boolean formulas of the variables $x_1, x_2 \cdots x_n$ such that every function $f : B^n \to B$ is represented by a formula $\phi_f \in E$. The set E is said to *fully represent* the set $B^n \to B$.

Definition 5. The *logical negation* of a function $g : B^n \to B$ is a function $\overline{g} : B^n \to B$ such that, on every assignment to the variables $x_1, x_2 \cdots x_n$,

$$\overline{g}(x_1, x_2 \cdots x_n) = \overline{g(x_1, x_2 \cdots x_n)}.$$

The logical negation of a function g is denoted by $\overline{g}$. For any function $g : B^n \to B$ and for every assignment, g and $\overline{g}$ must evaluate to different values: one of them must evaluate to $false$ and the other to $true$.

Definition 6. A run $\mathrm{P}(\phi_1, \phi_2)$ of a RAM program P is said to *decide the satisfiability of* $\phi_1 \wedge \phi_2$, or to *decide whether* $\phi_1 \wedge \phi_2$ *is satisfiable*, if and only if the run accepts its input (i.e., halts with the answer yes in R_1) if $\phi_1 \wedge \phi_2$ is satisfiable and rejects its input (i.e., halts with the answer no in R_1) otherwise. A run $\mathrm{P}(\phi_1, \phi_2)$ is said to *decide the falsifiability of* $\phi_1 \vee \phi_2$, or to *decide whether* $\phi_1 \vee \phi_2$ *is falsifiable*, if and only if the run accepts its input if $\phi_1 \vee \phi_2$ is falsifiable and rejects its input otherwise.

Theorem 1. Let E be a full representation of the set $B^n \to B$. There **does not exist** a RAM program P such that, for every pair of formulas $\phi_1, \phi_2 \in E$, $\mathrm{P}(\phi_1, \phi_2)$ correctly decides whether $\phi_1 \wedge \phi_2$ is satisfiable by following a terminated path that includes fewer than 2^n conditional jump instructions.

Proof. We will prove the theorem by contradiction. We first assume that there exists a RAM program P such that, for every pair of formulas $\phi_1, \phi_2 \in E$, P(ϕ_1, ϕ_2) correctly decides the satisfiability of $\phi_1 \wedge \phi_2$ by following a terminated path that includes fewer than 2^n conditional jump instructions. In other words, P(ϕ_1, ϕ_2) will accept the input if $\phi_1 \wedge \phi_2$ is satisfiable and reject the input otherwise, and P(ϕ_1, ϕ_2) will do so by following a terminated path that includes fewer than 2^n conditional jump instructions. The rest of this proof will derive a contradiction to this assumption.

Since E fully represents the set $B^n \rightarrow B$, every function $f : B^n \rightarrow B$ and its logical negation $\overline{f} : B^n \rightarrow B$ are represented by some Boolean formulas ϕ_f, $\phi_{\overline{f}} \in E$. Let S be a set containing, for each distinct function $f : B^n \rightarrow B$, one run P(ϕ_f, $\phi_{\overline{f}}$). In other words, for each function $f : B^n \rightarrow B$, S contains the run P(ϕ_f, $\phi_{\overline{f}}$), which is to decide whether the formula $\phi_f \wedge \phi_{\overline{f}}$ is satisfiable. Since there are $F = 2^{2^n}$ distinct functions in the set $B^n \rightarrow B$, the set S has F runs of the program P.

The set S may seem expensive to implement in terms of computing resources. However, S will only be used to prove that logically the RAM program P does not exist. An actual implementation of S is not needed.

Since, for every function $f : B^n \rightarrow B$ and for every assignment, either the function f or its logical negation $\overline{f}$ evaluates to $false$, and since ϕ_f, $\phi_{\overline{f}} \in E$ represent f and $\overline{f}$, for every assignment either ϕ_f or $\phi_{\overline{f}}$ evaluates $false$. Therefore, the formula $\phi_f \wedge \phi_{\overline{f}}$ is $false$ for every assignment and, thus, is not satisfiable. Hence, every run in the set S must eventually reject its input. By our assumption on P, every run in S must follow a terminated path that includes $2^n - 1$ or fewer conditional jump instructions and reject its input.

By Lemma 1, there are no more than 2^m terminated paths with each including m or fewer conditional jump instructions. Since each run in S follows a terminated path that includes $2^n - 1$ or fewer conditional jump instructions, by Lemma 1 there are no more than the following number of terminated paths that the runs in S may follow.

$$2^{(2^n-1)} = 2^{2^n}2^{-1} = F2^{-1} = F/2$$

To summarize, each of the $F = 2^{2^n}$ runs in the set S follows a terminated path that includes $2^n - 1$ or fewer conditional jump instructions to reject its input, but there are no more than $F/2$ such paths. Therefore, there is at least one such path that multiple runs in S follow. Let U be a path that multiple runs in S follow and let P(ϕ_g, $\phi_{\overline{g}}$) and P(ϕ_h, $\phi_{\overline{h}}$) be two runs in S that follow the path U. Since S contains one run of P for each distinct Boolean function of n variables, g and h must be different functions. Since, as discussed previously, all runs in S must reject their inputs, both P(ϕ_g, $\phi_{\overline{g}}$) and P(ϕ_h, $\phi_{\overline{h}}$) must reject their inputs. By Lemma 2, two other runs, P(ϕ_g, $\phi_{\overline{h}}$) and P(ϕ_h, $\phi_{\overline{g}}$), which are not in S, must follow the same path U and **reject** their inputs, as the two runs P(ϕ_g, $\phi_{\overline{g}}$) and P(ϕ_h, $\phi_{\overline{h}}$) do.

Now let us derive a contradiction to the assumption that the RAM program P exists. Since g and h are different Boolean functions, there exists an assignment s that makes g and h evaluate to different values. Hence, the assignment s will make g and $\overline{h}$ evaluate to the same value. If both g and $\overline{h}$ evaluate to $true$ on the assignment s, so will both of the formulas ϕ_g and $\phi_{\overline{h}}$, since ϕ_g, and $\phi_{\overline{h}}$ represent g and $\overline{h}$. Thus, $\phi_g \wedge \phi_{\overline{h}}$ is satisfiable. On

the other hand, if g and $\overline{h}$ evaluate to $false$ on the assignment s, then h and $\overline{g}$ will evaluate to $true$ on the assignment s and so will the formulas ϕ_h and $\phi_{\overline{g}}$, since ϕ_h and $\phi_{\overline{g}}$ represent h and $\overline{g}$. Thus, $\phi_h \wedge \phi_{\overline{g}}$ is satisfiable. Therefore, at least one of the two formulas $\phi_g \wedge \phi_{\overline{h}}$ and $\phi_h \wedge \phi_{\overline{g}}$ is satisfiable and, thereby, at least one of the two runs P(ϕ_g, $\phi_{\overline{h}}$) and P(ϕ_h, $\phi_{\overline{g}}$) **should accept** its input. However, as discussed previously, by Lemma 2 both P(ϕ_g, $\phi_{\overline{h}}$) and P(ϕ_h, $\phi_{\overline{g}}$) **reject** their inputs. That is, by Lemma 2, at least one of the two runs P(ϕ_g, $\phi_{\overline{h}}$) and P(ϕ_h, $\phi_{\overline{g}}$) **incorrectly** rejects its input. This contradicts our assumption that the program P exists such that, for every pair of formulas ϕ_1, $\phi_2 \in E$, P(ϕ_1, ϕ_2) **correctly** decides the satisfiability of $\phi_1 \wedge \phi_2$ by following a terminated path that includes fewer than 2^n conditional jump instructions. **Q.E.D.**

By Theorem 1, for any RAM program P, there is at least one pair of formulas ϕ_1 and ϕ_2 in any full representation of $B^n \rightarrow B$ such that P(ϕ_1, ϕ_2) cannot correctly decide the satisfiability of $\phi_1 \wedge \phi_2$ by executing fewer than 2^n conditional jump instructions. In other words, 2^n is a lower bound on the number of conditional jump instructions needed.

The proof for Theorem 1 does not rely on a specific representation of Boolean functions. Hence, the lower bound applies to the problem of deciding whether the conjunction of a pair of n-variable Boolean functions has a satisfying assignment, even if the two conjuncts are represented in the input as some expressions other than Boolean formulas of the form introduced previously.

Since there are many different Boolean formulas that represent a given Boolean function, there are many full representations of the set $B^n \rightarrow B$. Theorem 1 holds for any full representation E, even if E consists solely of minimized Boolean formulas that are derived by a Boolean minimization method.

Since the set $B^n \rightarrow B$ can be fully represented by a set of CNF formulas, the lower bound holds even if the conjuncts ϕ_1 and ϕ_2 are limited to CNF formulas.

Corollary 1.1. Let E be a set of CNF formulas that fully represents $B^n \rightarrow B$. There **does not exist** a RAM program P such that, for every pair of formulas ϕ_1, $\phi_2 \in E$, P(ϕ_1, ϕ_2) correctly decides whether the CNF formula $\phi_1 \wedge \phi_2$ is satisfiable by following a terminated path that includes fewer than 2^n conditional jump instructions.

By duality, Corollary 1.2 follows from Theorem 1.

Corollary 1.2. Let E be a full representation of $B^n \rightarrow B$. There **does not exist** a RAM program P such that, for every pair of formulas ϕ_1, $\phi_2 \in E$, P(ϕ_1, ϕ_2) correctly decides whether $\phi_1 \vee \phi_2$ is falsifiable by following a terminated path that includes fewer than 2^n conditional jump instructions.

By duality, Corollary 1.3 follows from Corollary 1.1.

Corollary 1.3. Let E be a set of DNF formulas that fully represents $B^n \rightarrow B$. There **does not exist** a RAM program P such that, for every pair of formulas ϕ_1, $\phi_2 \in E$, P(ϕ_1, ϕ_2) correctly decides whether the DNF formula $\phi_1 \vee \phi_2$ is falsifiable by following a terminated path that includes fewer than 2^n conditional jump instructions.

5 Restricted Formulas

Theorem 1 requires that the two conjuncts ϕ_1 and ϕ_2 be formulas from a full representation of $B^n \rightarrow B$. Since the following widely known sets of restricted formulas of n variables do not fully represent $B^n \rightarrow B$, Theorem 1 does not apply if the two conjuncts are limited to n-variable formulas from any of these sets: XOR-SAT, HORN-SAT, 2CNF, and 3CNF. Polynomial-time algorithms to decide 2CNF satisfiability, XOR-SAT, and HORN-SAT are known. The next theorem establishes a lower bound on conditional jump instructions for 3CNF satisfiability.

Definition 7. Let E_1 and E_2 be sets of Boolean formulas. A *satisfiability-preserving* mapping from E_1 to E_2 is a function $t : E_1 \rightarrow E_2$ such that, for every formula $\phi \in E_1$, the image $t(\phi) \in E_2$ is satisfiable if and only if ϕ is satisfiable. The function t is said to *preserve satisfiability*.

Definition 8. Let E_1 and E_2 be sets of Boolean formulas. A mapping from E_1 to E_2 that *preserves satisfiability over conjunction* is a function $t : E_1 \rightarrow E_2$ such that, for every pair of formulas ϕ_1, $\phi_2 \in E_1$, the formula $t(\phi_1) \wedge t(\phi_2)$ is satisfiable if and only if $\phi_1 \wedge \phi_2$ is satisfiable. The function t is said to be *satisfiability-preserving over conjunction*.

Definition 9. A set E of Boolean formulas is said to be a *satisfiability representation* of the set $B^n \rightarrow B$ if and only if there exist a full representation E_1 of the set $B^n \rightarrow B$ and a function $t : E_1 \rightarrow E$ that preserves satisfiability over conjunction. The set E is said to *satisfiability-represent* the set $B^n \rightarrow B$.

Theorem 2. Let E be a satisfiability representation of $B^n \rightarrow B$. There **does not exist** a RAM program P such that, for every pair of formulas ϕ_1, $\phi_2 \in E$, P(ϕ_1, ϕ_2) correctly decides whether the formula $\phi_1 \wedge \phi_2$ is satisfiable by following a terminated path that includes fewer than 2^n conditional jump instructions.

Proof. Our proof for Theorem 2 is essentially identical to that for Theorem 1, with the following adaptions:

1. The proof assumes that there exists a RAM program P such that, for every pair of formulas ϕ_1, $\phi_2 \in E$, P(ϕ_1, ϕ_2) correctly decides the satisfiability of $\phi_1 \wedge \phi_2$ by following a terminated path that includes fewer than 2^n conditional jump instructions.

2. Since E satisfiability-represents $B^n \rightarrow B$, there is a set E_1 that is a full representation of $B^n \rightarrow B$ and there is a function $t : E_1 \rightarrow E$ that preserves satisfiability over conjunction. Let the set S contain, for each function $f : B^n \rightarrow B$, one run P($t(\phi_f)$, $t(\phi_{\overline{f}})$), where ϕ_f, $\phi_{\overline{f}} \in E_1$ and, hence, $t(\phi_f)$, $t(\phi_{\overline{f}}) \in E$.

3. For every function $f : B^n \rightarrow B$ and for every assignment, one of f and $\overline{f}$ evaluates to $false$. Since ϕ_f and $\phi_{\overline{f}}$ represent f and $\overline{f}$, for every assignment one of ϕ_f and $\phi_{\overline{f}}$ evaluates to $false$. Hence, $\phi_f \wedge \phi_{\overline{f}}$ is $false$ for all assignments and, thus, is not satisfiable. Since t is satisfiability-preserving over conjunction, the formula $t(\phi_f) \wedge t(\phi_{\overline{f}})$ is not satisfiable. Hence, every run in S must eventually reject its input.

4. To derive a contradiction, let P($t(\phi_g)$, $t(\phi_{\overline{g}})$) and P($t(\phi_h)$, $t(\phi_{\overline{h}})$) be two runs in S that reject their inputs by following a common terminated path U that includes fewer than 2^n conditional jump instructions - as deptailed in the proof for Theorem 1, there must be at least two such runs in S. By Lemma 2, the two runs P($t(\phi_g)$, $t(\phi_{\overline{h}})$) and P($t(\phi_h)$, $t(\phi_{\overline{g}})$), which are not in S, must follow the same execution path U to reject their inputs, as the two runs P($t(\phi_g)$, $t(\phi_{\overline{g}})$) and P($t(\phi_h)$, $t(\phi_{\overline{h}})$) do. Since g and h are different Boolean functions, there exists an assignment s that makes g and h evaluate to different values. Therefore, g and $\overline{h}$ evaluate to the same value on the assignment s. If both g and $\overline{h}$ evaluate to *true* on the assignment s, then so will both of the formulas ϕ_g and $\phi_{\overline{h}}$ since ϕ_g and $\phi_{\overline{h}}$ represent g and $\overline{h}$. Hence, $\phi_g \wedge \phi_{\overline{h}}$ is satisfiable. Since t is satisfiability-preserving over conjunction, $t(\phi_g) \wedge t(\phi_{\overline{h}})$ is satisfiable too. On the other hand, if both g and $\overline{h}$ evaluate to $false$ on the assignment s, then both h and $\overline{g}$ evaluate to *true* on the assignment s, and the formula $t(\phi_h) \wedge t(\phi_{\overline{g}})$ can be similarly shown to be satisfiable. So, at least one of the formulas $t(\phi_g) \wedge t(\phi_{\overline{h}})$ and $t(\phi_h) \wedge t(\phi_{\overline{g}})$) is satisfiable. That is, at least one of the two runs P($t(\phi_g)$, $t(\phi_{\overline{h}})$) and P($t(\phi_h)$, $t(\phi_{\overline{g}})$) **should accept** its input. However, as discussed previously, by Lemma 2 both P($t(\phi_g)$, $t(\phi_{\overline{h}})$) and P($t(\phi_h)$, $t(\phi_{\overline{g}})$) **reject** their inputs. That is, by Lemma 2, at least one of the two runs P($t(\phi_g)$, $t(\phi_{\overline{h}})$) and P($t(\phi_h)$, $t(\phi_{\overline{g}})$) **incorrectly** rejects its input. This contradicts the assumption stated above in item 1. **Q.E.D.**

We give an example of a set of restricted Boolean formulas that satisfiability-represents $B^n \rightarrow B$. It is well known that the problem of CNF satisfiability can be reduced to 3CNF satisfiability, e.g., [2,7]. Specifically, when this reduction is applied to a CNF formula $C_1 \wedge C_2$, where C_1 and C_2 are CNF formulas, the reduction yields a formula $t(C_1) \wedge t(C_2)$ as the resultant 3CNF formula, where $t(C_1)$ and $t(C_2)$ are 3CNF formulas and are derived by applying the reduction to C_1 and C_2 respectively. The formulas $t(C_1)$ and $t(C_2)$ are satisfiable if and only if C_1 and C_2 are, respectively, and the resultant 3CNF formula $t(C_1) \wedge t(C_2)$ is satisfiable if and only if the original CNF formula $C_1 \wedge C_2$ is. This reduction introduces distinct new variables into the resultant 3CNF formulas. With the new variables being distinct, this reduction defines a mapping from CNF formulas to 3CNF formulas that is satisfiability-preserving over conjunction. Let E_1 be a set of CNF formulas that fully represents $B^n \rightarrow B$. This reduction can be used to transform each CNF formula in E_1 into a 3CNF formula. Let E be the set of the resultant 3CNF formulas. The set E satisfiability-represents the set $B^n \rightarrow B$.

The following corollary directly follows from Theorem 2.

Corollary 2.1. Let E be a set of 3CNF formulas that satisfiability-represents $B^n \rightarrow B$. There **does not exist** a RAM program P such that, for every pair of 3CNF formulas ϕ_1, $\phi_2 \in E$, P(ϕ_1, ϕ_2) correctly decides whether the 3CNF formula $\phi_1 \wedge \phi_2$ is satisfiable by following a terminated path that includes fewer than 2^n conditional jump instructions.

Similarly, there is a reduction from the problem of DNF falsifiability to 3DNF falsifiability [1]. By duality, the following corollary follows from Corollary 2.1. The term *falsifiability-represent* is the dual of the term satisfiability-represent defined previously. A detailed definition of the term falsifiability-represent parallels Definitions 8-9.

Corollary 2.2. Let E be a set of 3DNF formulas that falsifiability-represents $B^n \rightarrow B$. There **does not exist** a RAM program P such that, for every pair of 3DNF formulas ϕ_1, $\phi_2 \in E$, P(ϕ_1, ϕ_2) correctly decides whether the 3DNF formula $\phi_1 \vee \phi_2$ is falsifiable by following a terminated path that includes fewer than 2^n conditional jump instructions.

6 Lower Bounds on Two Other Machine Models

By similar proofs, we previously established in [3,4]

- 2^n as a lower bound [4] on the number of conditional branches that Post's Formulation 1 [6] needs to execute in order to decide Boolean satisfiability, and
- $2^n log_k 2$ as a lower bound [3] on the number of "moves" that a Turing machine [8] with k symbols in its tape alphabet needs to make in order to decide Boolean satisfiability. The lower bound is 2^n when $k = 2$, i.e., when a Turing machine uses a binary tape alphabet.

Unlike the RAM or Post's formulation 1, a Turing machine bundles each decision with a "move", which includes a state transition, a movement of the read/write head, and an operation to write a symbol to a tape square. Hence, the lower bound $2^n log_k 2$ is on the number of moves that a Turing machine makes, whereas in the context of a RAM program or Post's Formulation 1, the lower bound is on the number of conditional jump instructions and does not apply to other types of instructions that a RAM program or Post's Formulation 1 executes.

References

1. Cook, S.A. The complexity of theorem proving procedures. In *Proceedings, Third Annual ACM Symposium on the Theory of Computing* (1971), pp. 151-158.

2. Hopcroft, J.E., Ullman, J.D., *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

3. Hsieh, S. C. A Lower Bound for Boolean Satisfiability on Turing Machines. preprint (2014) available at *http://arxiv.org/abs/1406.5970*

4. Hsieh, S. C. A Lower Bound of 2^n Conditional Branches for Boolean Satisfiability on Post Machines. preprint (2014) available at *http://arxiv.org/abs/1406.6353*

5. Machtey, M., Young P., *An Introduction to the General Theory of Algorithms.* North-Holland, New York, NY, 1978.

6. Post, E.L. Finite Combinatory Processes-Formulation 1. *The Journal of Symbolic Logic* 1 (1936) pp. 103-105.

7. Sipser, M. *Introduction to the Theory of Computation.* 2nd ed. Thomson Course Technology, 2006.

8. Turing, A.M. On Computable Numbers, with an Application to the Entscheidungs problem. In *Proceedings of the London Mathematical Society* (1936), pp.230-265.