SAT SOLVER

April 6, 2016

Roll number of Students:-

140050037

140050043

140050059

140050067

Contents

INTRODUCTION

A hardware tool for checking satisfiability of a propositional logic formula given in a Conjunctive Normal Form i.e determines whether a given formula ϕ over prepositional variables p_i can be made true under any assignment of truth values to p_i.Validity checking is dual to satisfiability checking.validity of ϕ can be determined using a SAT solver by checking whether $\neg\phi$ is satisfiable and complementing the answer $VALIDITY(\phi) = \neg SAT(\neg\phi)$.

We will implement SAT solver using DPLL(Davis - Putnam - Logemann - Loveland) algorithm. It's a backtracking based search algorithm. The algorithm is non deterministic polynomial. There is no known algorithm that efficiently solves SAT and it is believed that no such algorithm exists. SAT occurs in practise in many fields like circuit solving, Artificial intelligence, Puzzle solving and many more areas.

Brief Algorithm

Algorithm 1 DPLL

Input: A set of clauses ϕ.
Output: A Truth Value.

```
1:  procedure DPLL
2:      if no clause is left to be satisfied then
3:          return satisfied
4:      if a clause is empty then
5:          return unsatisfied
6:      if a free variable x_i is positive everywhere then
7:          V(x_i)=T and remove all clauses containing x_i (φ')
8:          return DPLL (φ')
9:      if a free variable x_i is negative everywhere then
10:         V(x_i)=F and remove all clauses containing x_i (φ')
11:         return DPLL (φ')
12:     x_i ←choose a variable and assignment(φ)
13:     check whether assignment results in conflict
14:     if conflict arises then change assignment and now check again for conflict then
15:         if again conflicts arises then
16:             return unsatisfiable
17:     now update formula φ (φ')
18:     return DPLL (φ')
```

Explanation

Preposition variables are denoted by $x_1, x_2, ..., x_n$. They assigned truth values True (denoted by T) and False (denoted by F). The truth value assigned to a variable x is denoted by V(x). A variable is said to be free if it has not been assigned a truth value yet. A literal l is either a variable x_i or its negation $\neg x_i$. A clause C_i is collection of literals joined by disjunction. A formula ϕ is collection of clauses joined by conjunction. A clause is said to be satisfied if at least one of its literals assumes value 1, unsatisfied if all of its literals assume value 0. A formula is said to be satisfied if all its clauses are satisfied, and is unsatisfied if at least one clause is unsatisfied.

Heuristics for selecting variable

A crucial factor influencing the performance of the DPLL-based SAT solver is its decision heuristic i.e which variable to use. This heuristic decides which variable to choose at each decision point during the search and what value to assign it first. For deciding variable we will be using Maximum Occurrences on clauses of Minimum Size(MOM's) heuristic.

MOM's Heuristic

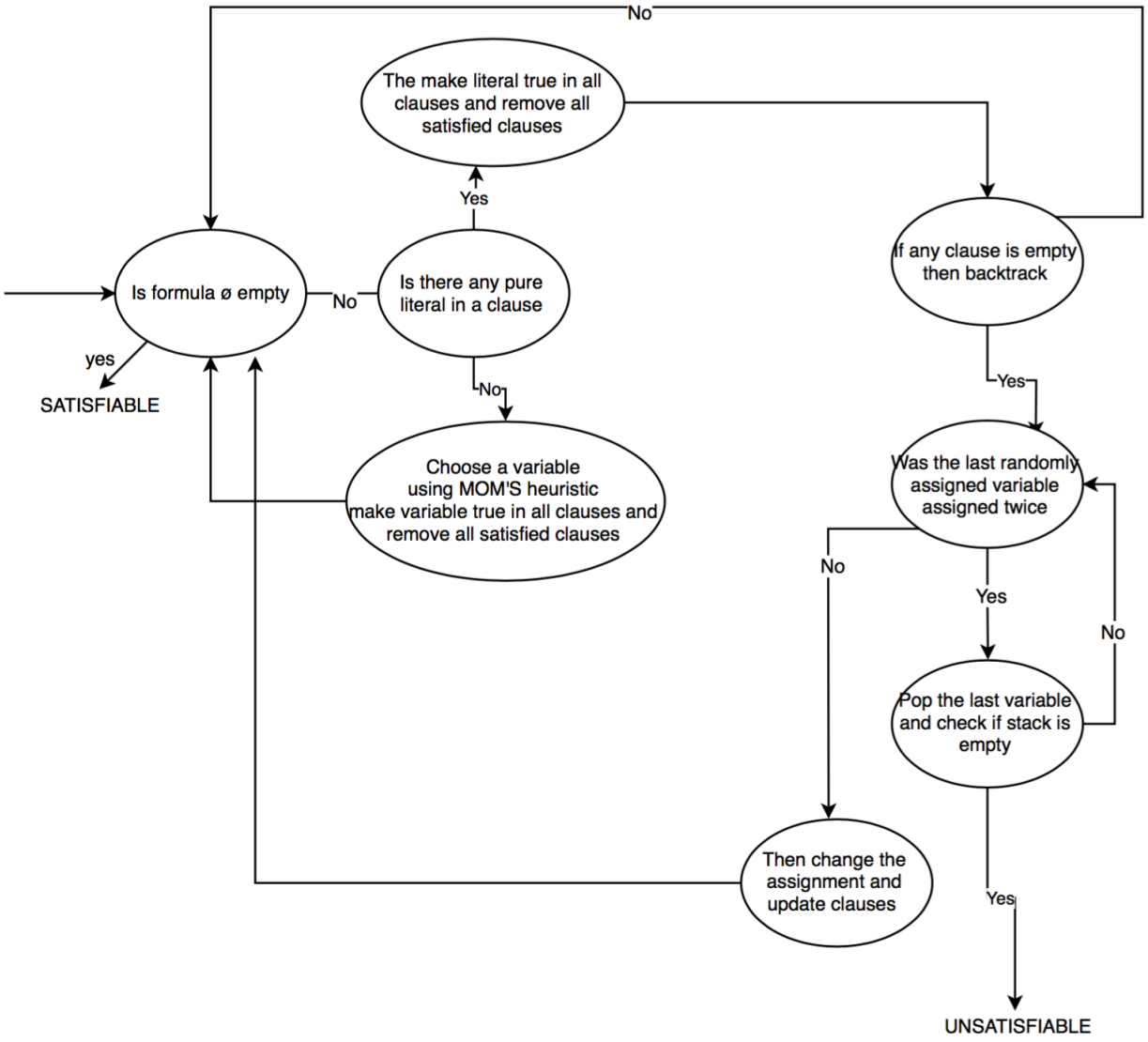
Intuition behind MOM's heuristic is giving preference to those variables which have maximum frequency in the clauses having minimum size. If more than one variable has same maximum frequency then repeat the same method for clauses having the second minimum size.
The empirical formula used to choose the variable is the one that maximizes the function.

$$[\mathbf{f}^*(\mathbf{x}) + \mathbf{f}^*(\neg\mathbf{x})] * \mathbf{2^k} + \mathbf{f}^*(\mathbf{x}) * \mathbf{f}^*(\neg\mathbf{x})$$

where $f^*(x)$ is the number of occurrences of a literal x in the smallest non-satisfied clauses and k is sufficiently chosen to be large.

To the selected variable assign value true if the variable appears in more smallest clauses as a positive literal, and value false otherwise.

State Diagram



HIGH LEVEL ARCHITECTURE

As a first step converting recursive algorithm stated above into iterative form for easy hardware implementation.

Algorithm

Algorithm 2 DPLL Iterative

Input: A set of clauses ϕ.
Output: A Truth Value.

```
1:  procedure DPLL
2:      stack:- randomDecision,allAssignments
3:      while true do
4:          if no clause is left to be satisfied then return satisfied
5:          if a clause is empty  then
6:              while !(top of randomDecision stack has been assigned twice) do
7:                  Let x_i = top of randomStack
8:                  do
9:                      remove the last set of clauses which have been stored
10:                     remove top from allAssignents stack
11:                 while removed element != x_i
12:             change the truth value of x_i and push x_i into both stacks
13:             update the clauses and store it
14:             remove the clauses that become true
15:         if x_i is a free variable or a part of unit clause then
16:             update the clauses and store the present values
17:             assign value of x_i appropriately
18:             remove all clauses containing that become true
19:             push x_i into allAssignments stack
20:             update φ
21:         x_i ←choose a variable and assignment(φ)
22:         push x_i on both stacks
23:         update the set of
24:         update φ
25:         if top of allAssignments stack is empty then
26:             return unsatisfied
```

Explanation of algorithm

Variables used:

The randomDecision stack consists of all the variables for which the assignments were done based on the heuristic which we have used in the algorithm.

The allAssignments stack consists of all the variables who's value has already been assigned based on either the heuristic or because of it being a free variable or a part of a unit clause. i.e it contains all the decisions(assignments) which we taken so far in the run.

Termination Condition:

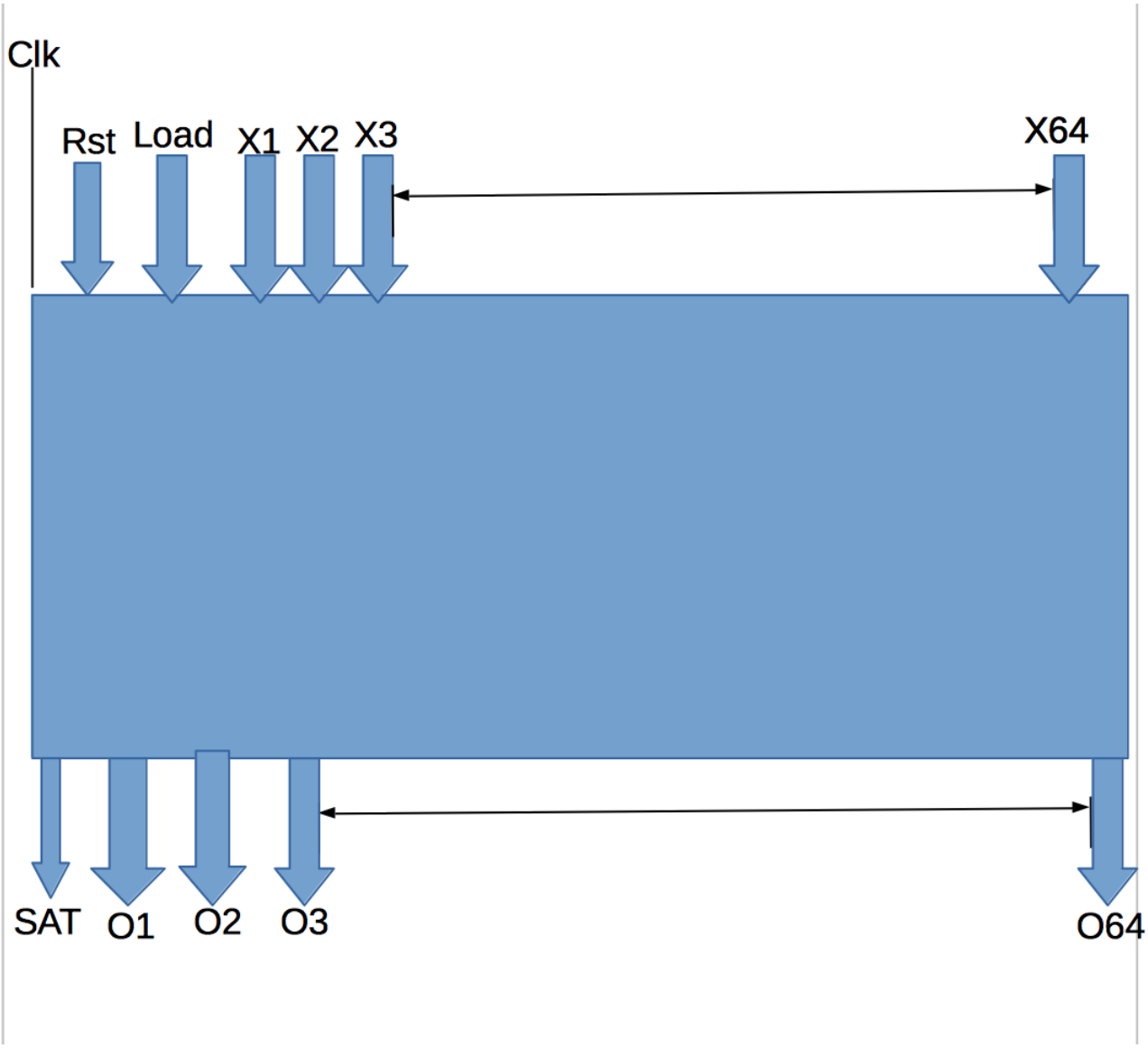
In case of satisfiability, i.e if the formula given is satisfiable, all the set of clauses yet to be satisfied becomes empty.

In case the formula is unsatisfiable, the random decision stack becomes empty.

Procedure

As long as the variable(say x_i) on top of the randomDecision stack has not been assigned any value twice, we can choose an assignment for it. Now because of this assignment some clauses become satisfied and in some clauses where the literal with xi evaluates to false. Remove the variable from the clause or the clause containing the variable itself as is needed. If a conflict arises backtrack (while also undoing the assignments) till there is a variable in the randomDecision stack that was not yet assigned both the choices and assign the other choice to it

Data Path



DESCRIPTIONS OF FUNCTIONALITIES OF BLOCK

There is a single block. It takes an input of 64 variables as clauses in the form as described in the problem statement.
If the case is UNSAT, then SAT output will be 0 and other outputs are dont cares otherwisse SAT i 1 and each of the remaining 64 outputs represent one of the satisfiable assignments for the given CNF formula

ASSUMPTIONS

Input should not have empty clause.
The circuit is assumed to have 65 outputs.
We assume that none of the input variables can not have 1 and 1.
We also assume the load to be zero untill the circuit completes the calculations.
Once load turns high from low we are assuming that it stays high for atleast two or more even cycles and after that if it becomes zero then it should not be immediately followed by a 1.

HOW WE TESTED OUR PROGRAM

We looked at functionalities of each block of code and created a case to verify that the block is indeed working properly. We also generated some random test cases and tested the code against each one of those. A brief description of each of the test cases is provided in the test bench.

HOW WE SHARED THE WORK BETWEEN GROUP MEMBERS

Backtracking and input are done by 140050037 and 140050043 single literal case and heuristic and unsat case is done by 1400050059 and 140050067

INSTRUCTIONS TO RUN THE PROGRAM

Open the file sat_solver.xise in Xilinx ISE and run behavorial check syntax and simulate using any of the test benches provided.

REFERENCES

Branching heuristics from this link
http://research.cs.wisc.edu/wpis/papers/tr1699.pdf
For basic introduction to problem
https://en.wikipedia.org/wiki/Boolean_satisfiability_problem
https://en.wikipedia.org/wiki/DPLL_algorithm
For drawing state diagram
https://www.draw.io/
For drawing Data path
https://www.gliffy.com/
For a good understanding of DPLL algorithm
http://resources.mpi-inf.mpg.de/departments/rg1/teaching/autrea-ss10/script/lecture4.pdf
http://www.inf.ed.ac.uk/teaching/courses/propm/papers/main_lan1.pdf