

# Multivalued Boolean Satisfiability: MV-SAT

Mel Tsai

[mtsai@eecs.berkeley.edu](mailto:mtsai@eecs.berkeley.edu)

10/8/2002

## Overview

- Introduction and Motivation
- Problem Formulation
- MV-SAT Algorithm
- Software Implementation
- Results
- Conclusion and Future Work

## Introduction

- “Regular” SAT: Propositional Satisfiability
  - Find an assignment to the variables of a boolean function  $f$  such that  $f$  evaluates to true (1).
  - Example:  $f = (\bar{x} + y)(y + \bar{z})(x + z)$ . A satisfying assignment is  $y = 1, z = 1, x = \text{don't care}$ .
- Propositional Satisfiability has many applications in EDA.
  - ATPG
  - Logic Synthesis
  - Verification
  - etc.

## MV-SAT: Motivation

- SIS is being extended into the multivalued domain.
- How to extend boolean satisfiability to the multivalued case?
- Is there an inherent advantage to “multivalued” SAT or will conversion to binary always be the most efficient?
- Project Goal: Write an MV-SAT solver and see if idea is useful.

## Problem Formulation

- MV-SAT is similar to propositional SAT
  - Defined over a set of clauses  $C$  and input MV-literals  $X$
  - Example:
 
$$(X^{\{2,4\}} + Y^{\{3,4\}})(X^{\{3,8\}} + Z^{\{0,1,10\}})(Y^{\{4,5\}} + Z^{\{0,10\}})$$
  - One solution to the above is  $X = 2, Z = 1, Y = 5$ . This allows the function to evaluate to “true”.
- Can we use traditional SAT techniques and heuristics in the multivalued case?

## MV-SAT Algorithm

- “Unateness” idea in MV-SAT:

$$(X^{\{0,1\}} + Y^{\{1,3\}})(X^{\{1,2\}} + Z^{\{0,1,10\}})(X^{\{0,3\}} + Z^{\{0,10\}})$$

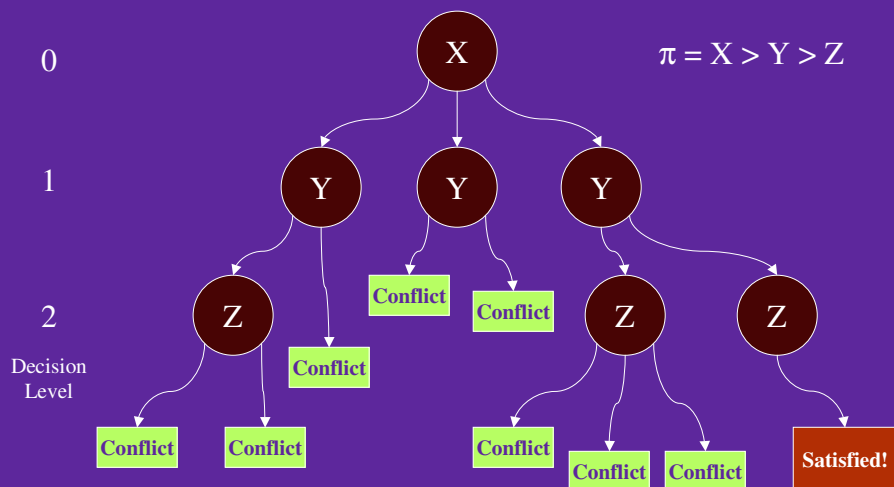
3	2	1	0
0	0	1	1
0	1	1	0
1	0	0	1

- In positional notation, column with most “1”s is the most unate value for that variable.

## MV-SAT: Algorithm (cont.)

- “Strong Unateness” in MV-SAT: Column of all 1’s. Analogous to the pure literal rule in SAT.
- Algorithm uses the same ideas as Davis-Putnam (ca. 1960) for boolean SAT:
  - Find strongly unate variables and fix their values
  - Start with most frequently occurring variable and branch according to most unate value for the variable.
  - Go to next most frequently occurring variable and continue until conflict arises
  - If conflict, undo last decision and keep going.

## MV-SAT: Algorithm (cont.)



## MV-SAT: Algorithm (cont.)

- Non-Chronological Backtracking
  - Backtracking to decision level  $n-2$  or earlier instead of just simple  $n-1$  backtracking.
  - Algorithm “looks back” to see original cause of conflict
- Conflict Analysis
  - As conflicts arise, add new clauses (“primes”) to the clause database so that common conflicts can be avoided earlier in the decision tree.

## Software Implementation

- Written in ANSI C, compiled under Linux
- Currently ~2500 lines of code
- Input file format:

```
// Input file for MV-SAT
num_variables
num_clauses

num_variables_in_clause_1, var_num, var_value, var_num, var_value...
num_variables_in_clause_2, var_num, var_value, var_num, var_value...
num_variables_in_clause_3, var_num, var_value, var_num, var_value...
.
.
.
num_variables_in_clause_n, var_num, var_value, var_num, var_value
```

## Software Implementation

### ● Current Features:

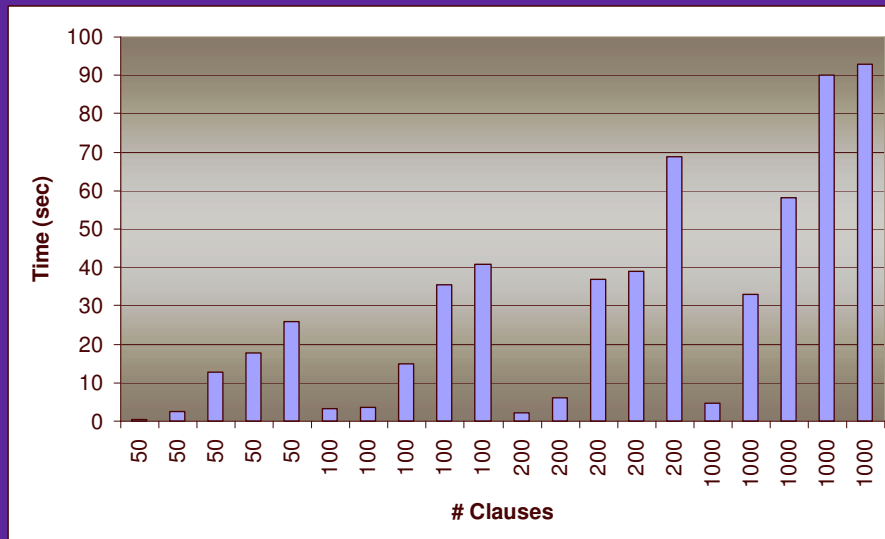
- “Exhaustive Search” Testing
  - Used for initial testing of clause database
  - Impractical for #variables > 20
- Heuristics-based MV-SAT:
  - Written with recursive functions
  - Literal sorting and reordering
  - Simple backtracking
  - Allows user to set a timeout limit or #backtracks limit for large input files

## Algorithm Difficulties

### ● Things that are “ugly” in software:

- Calculating unateness of a particular variable assignment
- Continually resorting remaining variable space after each decision level
- Keeping track of clause (un)satisfaction
- Keeping clause database small enough to remain in cache: Tradeoff between keeping a lot of useful information in the clause database vs. traversing the database quickly

## Initial Results



## Preliminary Conclusions

- Right now it is difficult to gauge efficiency of MV-SAT... Need more testing!
  - Need a MV-SAT to CNF file converter, test using GRASP
  - Run MV-SAT on the DIMACS test suite... How well does MV-SAT perform in binary?
  - How to write a better random input file generator... Generate hard instances MV-SAT?
  - Test MV-SAT using real life problems
- Unfortunately, it seems as though MV-SAT may never be faster than conversion to binary
  - Too much overhead for MV!

## Future Work

- Current MV-SAT seems slow... Many new ideas for improvement
  - Reorganization of the clause database
  - Write using non-recursive functions
  - Implement non-chronological backtracking and conflict analysis
  - MV-SAT could benefit greatly from hardware acceleration