Boolean Satisfiability: The Central Problem of Computation

Peter Kogge

(p. 299) SAT: Boolean Satisfiability

- **wff**: well-formed-formula constructed from
  - A set V of Boolean variables
  - Boolean operations AND, OR, NOT
- **Satisfiability**: is there a substitution of 0s and 1s to variables that makes the wff true
  - i.e. makes all clauses simultaneously true
- **Unsatisfiability**: no substitution makes all clauses true at same time
- See references in “Links” class page

CNF: Clausal Normal Form

- wff restructured as AND of a set of clauses
  - Each clause an OR of a set of literals
  - Each literal a variable or its negation
- For a wff in clausal form to be true
  - All clauses must be true
  - For any clause to be true at least one literal must be true
- Example: (~x v y) & (x v y) & (x v ~y)
  - x=1, y=1 makes expression true
- (~x v y) & (x v y) & (x v ~y) & (~x v ~y)
  - No assignment of values make this true

Why Does SAT Matter

- Huge range of direct applications
- Will show that ***ALL*** computable functions can be converted into a SAT problem
- If we can solve SAT quickly, we can solve ***any*** computable problem quickly
- But ***no one*** has been able to find such a solution!

Applications

Following list taken from http://logos.ucd.ie/~jpms/talks/talksite/jpms-wodes08.pdf

- **Circuit construction and simulation**
- **Model checking: H/W, S/W, test patterns**
- **AI: Planning; Knowledge representation; Games**
- **Bioinformatics: Haplotype inference; Pedigree checking; Maximum quartet consistency; etc.**
- **Design automation:**
- **Equivalence checking; Delay computation; Fault diagnosis; Noise analysis; etc.**
- **Security: Cryptanalysis; Inversion attacks on hash functions; etc.**
- **Computationally hard problems: Graph coloring; Traveling salesperson; etc.**
- **Mathematical problems: van der Waerden numbers; etc**
- **Core engine for many other problem domains**

SAT Problem Sizes

- Hundreds of thousands to millions of variables
- Huge numbers of clauses
- Often very large numbers of literals per clause
- Sample problem sources:
  - http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html
- There is even a yearly competition that has been going on for decades
  - Current 2017: https://baldur.iti.kit.edu/sat-competition-2017/index.php?cat=certificates
  - 2016: https://baldur.iti.kit.edu/sat-competition-2016/index.php?cat=certificates

Example: Sudoku to SAT

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Fill in all blanks so 1…9 appear on every row, column, and 3x3 grid

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

- **Define 729 variables $x_{i,j,d}$ ($1 \le i,j,d \le 9$) such that**
  - **$x_{i,j,d}$ = 1 if cell (i,j) has digit d, 0 otherwise**
- **81 clauses: 1 for each cell (i,j) to ensure it has a digit:**
  - **$(x_{i,j,1} \vee x_{i,j,2} \vee \ldots x_{i,j,9})$**
- **81 sets of 36 clauses to ensure no cell has 2 digits:**
  - **For each of $1 \le d < d' \le 9$: $(\sim x_{i,j,d} \vee \sim x_{i,j,d'})$**
- **To state that row i, for example, has all 9 digits:**
  - **AND of 9 clauses (1 for each value of d) where d'th clause is $(x_{i,1,d} \vee \ldots \vee x_{i,9,d})$**
  - **And 9 sets of 36 = 324 clauses to ensure uniqueness $(\sim x_{i,j,d} \vee \sim x_{i,j',d})$**
- **Repeat construction for all rows, columns, grids**
- **Total of 11,745 clauses (most with 2 literals/clause, rest have 9)**
- **Initialize cells by setting certain variables, e.g. $x_{1,1,5}$=1 and $x_{1,1,d}$ = 0 for $d \neq 5$**

A 2x2 Sudoku

1	

1	2
2	1

- 8 variables: $x_{1,1,1}$, $x_{1,1,2}$, $x_{1,2,1}$, $x_{1,2,2}$, $x_{2,1,1}$, $x_{2,1,2}$, $x_{2,2,1}$, $x_{2,2,2}$
- 4 clauses to ensure a digit/cell:
  - $(x_{1,1,1} \vee x_{1,1,2})\&(x_{1,2,1} \vee x_{1,2,2})\&(x_{2,1,1} \vee x_{2,1,2})\&(x_{2,2,1} \vee x_{2,2,2})$
- 4 sets of 1 clause to ensure no duplicates:
  - $(\sim x_{1,1,1} \vee \sim x_{1,1,2})\&(\sim x_{1,2,1} \vee \sim x_{1,2,2})\&(\sim x_{2,1,1} \vee \sim x_{2,1,2})\&(\sim x_{2,2,1} \vee \sim x_{2,2,2})$
- 4 clauses for row 1:
  - $(x_{1,1,1} \vee x_{1,2,1})\&(x_{1,1,2} \vee x_{1,2,2})\&(\sim x_{1,1,1} \vee \sim x_{1,2,1})\&(\sim x_{1,1,2} \vee \sim x_{1,2,2})$
- 4 clauses for row 2:
  - $(x_{2,1,1} \vee x_{2,2,1})\&(x_{2,1,2} \vee x_{2,2,2})\&(\sim x_{2,1,1} \vee \sim x_{2,2,1})\&(\sim x_{2,1,2} \vee \sim x_{2,2,2})$
- 4 clauses for column 1:
  - $(x_{1,1,1} \vee x_{2,1,1})\&(x_{1,1,2} \vee x_{2,1,2})\&(\sim x_{1,1,1} \vee \sim x_{2,1,1})\&(\sim x_{1,1,2} \vee \sim x_{2,1,2})$
- 4 clauses 4 column 2:
  - $(x_{1,2,1} \vee x_{2,2,1})\&(x_{1,2,2} \vee x_{2,2,2})\ \&(\sim x_{1,2,1} \vee \sim x_{2,2,1})\&(\sim x_{1,2,2} \vee \sim x_{2,2,2})$
- 2 Initialization clauses: $x_{1,1,1}$ & $\sim x_{1,1,2}$

Variants of SAT in CNF

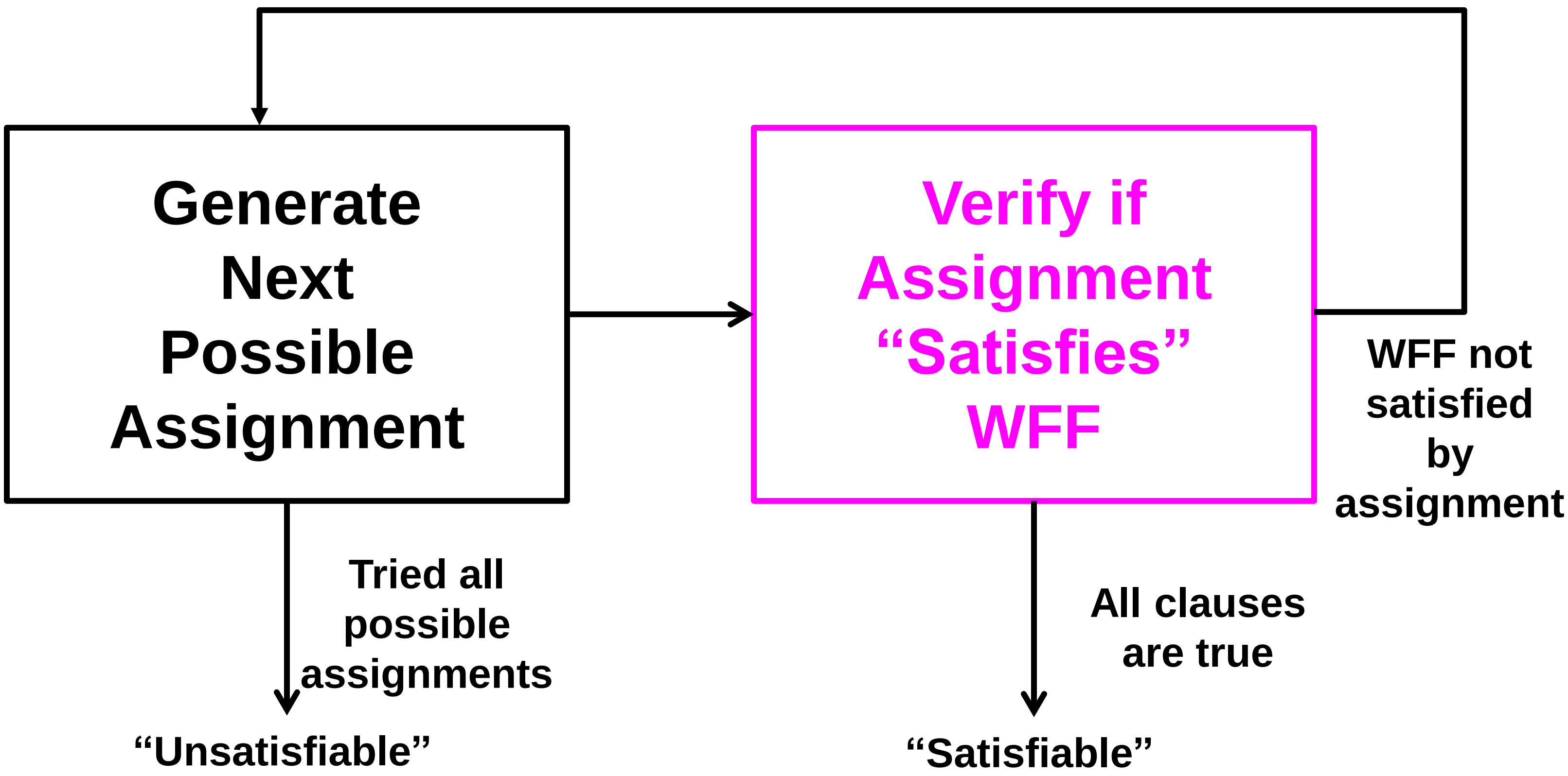
- **1-SAT**: all clauses have <u>exactly</u> 1 literal
  - Each clause is one literal
  - If any 2 clauses are a variable & its complement, then reject
  - E.g. x_1 & x_2 & ~x_3 satisfied by x_1 =1,x_2 =1, x_3 =0
  - But add on clause ~x_1 and unsatisfiable
- **2-SAT**: all clauses have *at most* 2 literals
  - Clause: (L_{i1} V L_{i2})
- **3-SAT**: all clauses have *at most* 3 literals
  - Clause: (L_{i1} V L_{i2} V L_{i3})
  - At least one literal in each clause must be true

The Simplest SAT Solver

- Generate all 2^V assignments to V variables
- For each assignment, check each clause
- Satisfiable: Some assignment makes all clauses true
- Unsatisfiable: no assignment works

x	y	z	x V ~y	y V z	~xV~z	~xV~yVz	xVyV~z	All Clauses	All but last
0	0	0	1	0	1	1	1	0	0
0	0	1	1	1	1	1	0	0	1
0	1	0	0	1	1	1	1	0	0
0	1	1	0	1	1	1	1	0	0
1	0	0	1	0	1	1	1	0	0
1	0	1	1	1	0	1	1	0	0
1	1	0	1	1	1	0	1	0	0
1	1	1	1	1	0	1	1	0	0

Brute Force Approach



Brute Force Algorithm

for each combination of variable values

for each clause in wff

for each literal in clause

look up variable in assignment

if literal is true: break to next clause

if all literals are false:

break to next combination

if all clauses are true: break “Satisfiable”

if no combination satisfied: “Unsatisfiable”

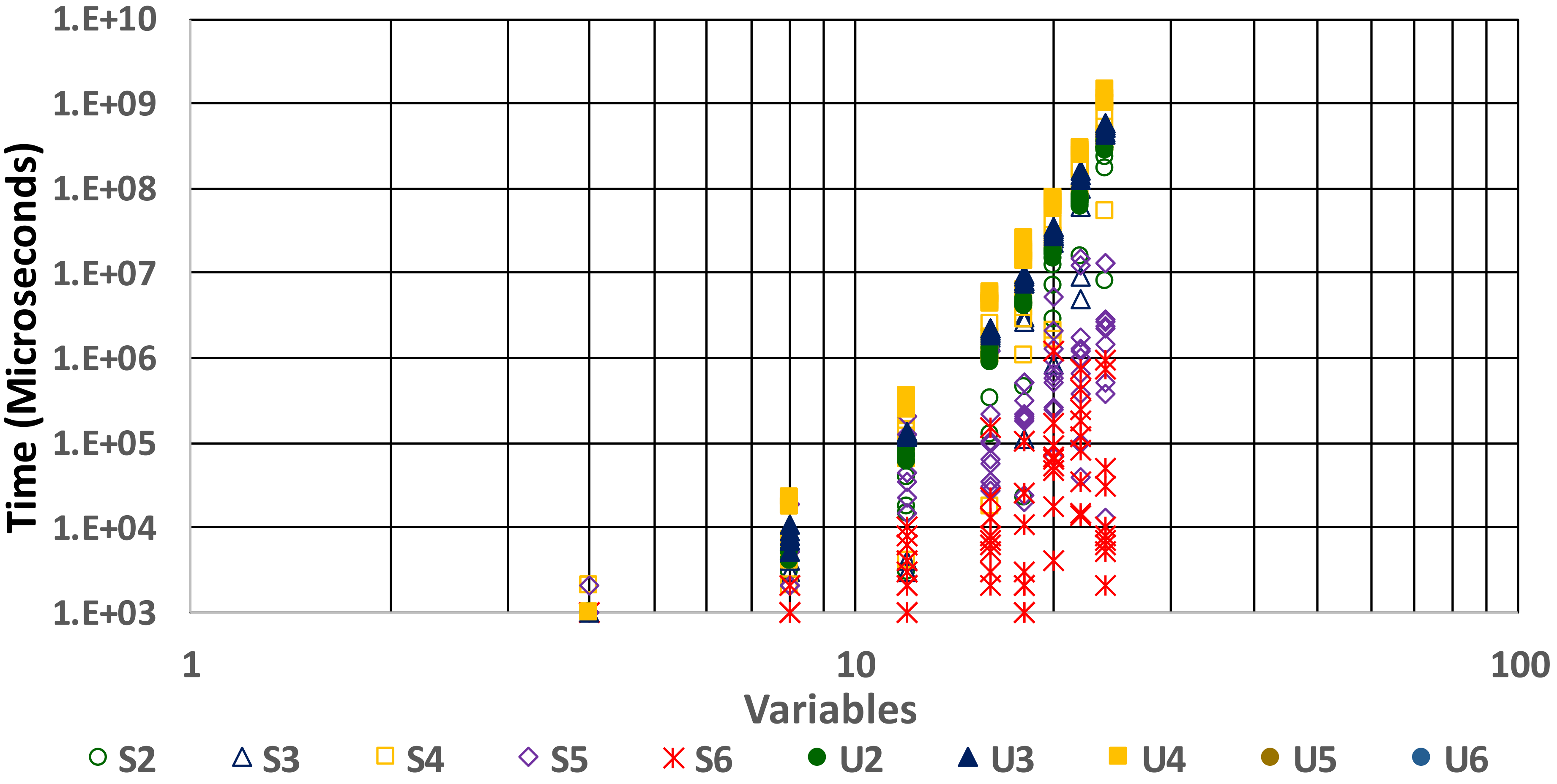
Verifier

2^V C K

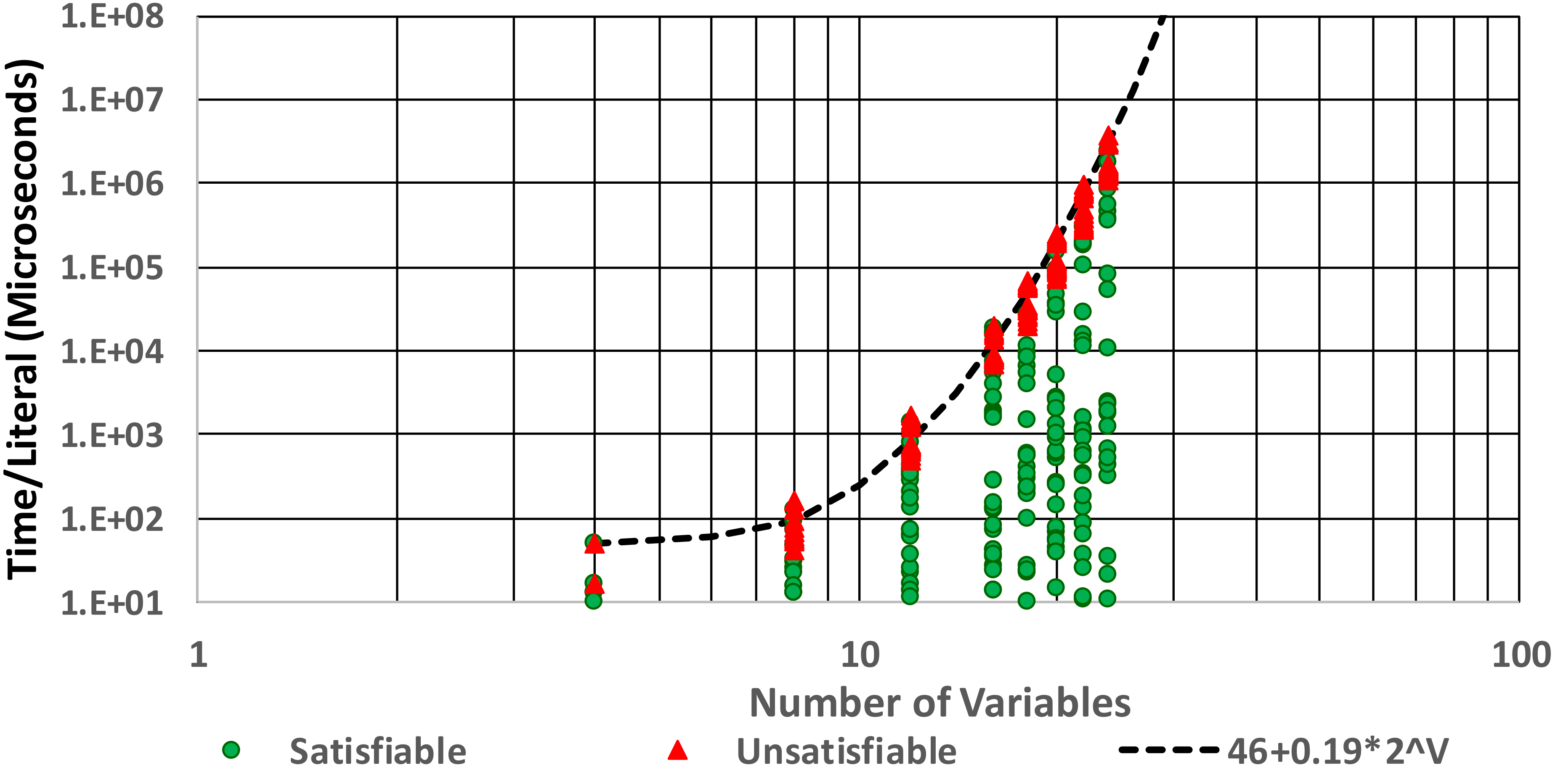
Time Complexity: $O(2^V * C * K)$

- V = # variables
- C = # Clauses
- K = # Literals per Clause

A Python Implementation



Dividing by CK=# Literals



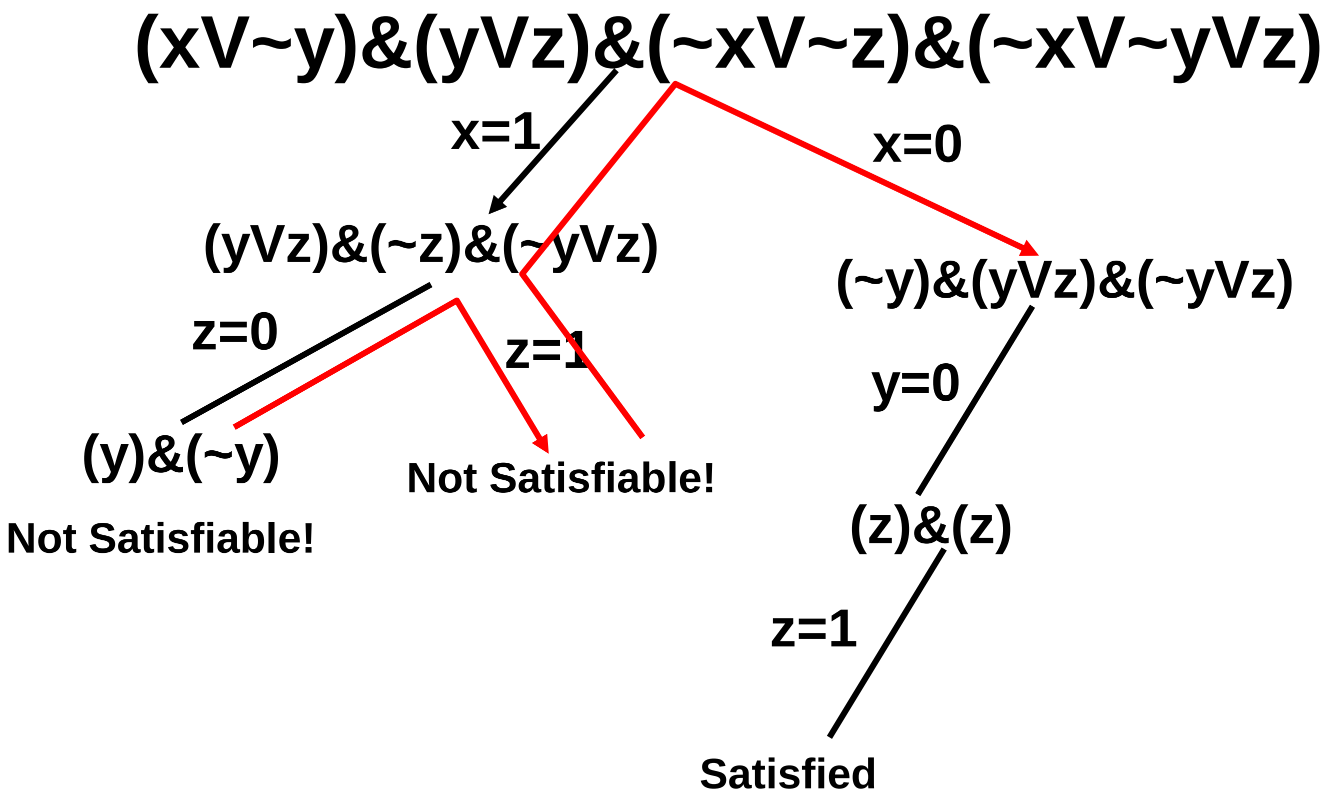
Backtracking: Core to Real Solvers

- Consider “incremental” approach that generates assignment dynamically
- Keep track of state of clauses under current partial assignment; clauses may be
  - **True**: some literal in clause has a variable value that makes it true
  - **False**: all literals in clause have variable values that make literals false
  - **Undetermined**: one or more literals have variables without any current assigned value
- Keep “stack” of order of assignments to allow backtrack if current assignment doesn’t work

Basic Backtracking

- Select some variable from a indeterminate clause
- Select value to give to that variable (to make some clause true)
  - Save (on stack) variable and value as a "CHOICE POINT"
- Ignore all clauses now true
- If no clause remains, declare "Satisfied"
  - Values on stack are satisfying assignment
- If some clause is now "false":
  - Go to top choice point, reverse value and try again
  - If top variable has tried both values, pop choice point, and repeat on choice point below below
  - If stack is now empty, declare "Unsatisfiable"
- If no clauses false and some still undetermined, repeat above on a different variable that has no value

Equivalent to a "Tree Traversal"



Red: Backtrack to last Choice Point and try another

Another Example

(xV~y)&(yVz)&(~xV~z)&(~xV~yVz)&(xVyV~z)

The Unit Clause Rule

- Additional trick: When a clause has *only one* undetermined literal
  - Add a choice point entry with that variable
  - Assign value to variable to make literal true
  - With flag that reversing value need not be tried
- Many other heuristics have been developed
- Average complexity *greatly* reduced
- But for kSAT, k>2, worst case still $O(2^V)$

Special Case: 2SAT

- **Speedup observation:**
  - **Assume we guess x_i = 1 (build a choice point)**
  - **All clauses with x_i as a literal are now true**
- **Now look at all clauses of form (~x_i V L_j)**
  - **~x_i is false from assignment**
  - **so L_i *must be* true => *new assignment***
  - **Can repeat as long as we generate new assignments**
- **Backtrack when we get conflicting assignments to same variable**
- **Variations are *polynomial* even in worst case**
  - **Possible to get linear time**

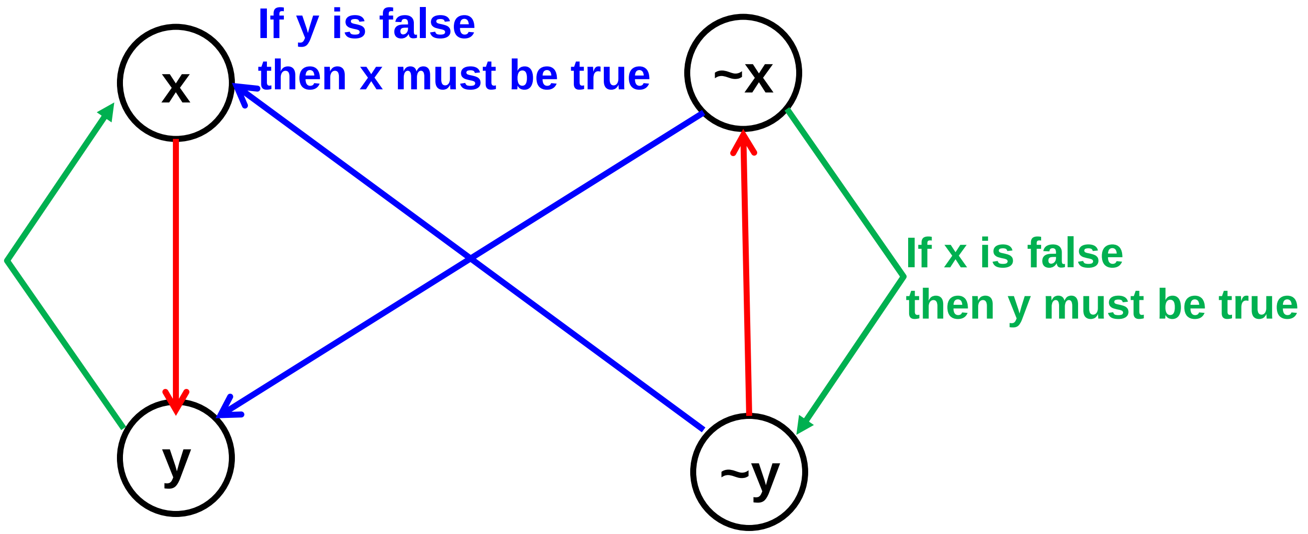
Alternative 2SAT Graph Algorithm

- If V variables, generate 2V vertices
  - pairs labelled x_i and $\sim x_i$
- For each clause (L_i V L_k) using variables x_i and x_k, generate 2 edges in the graph
  - $\sim L_i$ to L_k
  - $\sim L_k$ to L_i
- Unsatisfiable if for any x_i there is a path
  - from x_i to $\sim x_i$
  - and $\sim x_i$ to x_i
- Satisfiable if no such path

2SAT as Domino Chains

Example:

(~x V y) & (x V y) & (x V ~y)



What happens when we add clause (~x V ~y)?

Your Turn: Bipartite Matching

- What are variables?
- How to guarantee at least one match per vertex?
- How to guarantee only 1 match per vertx?