Foundations and Theoretical Aspects of Propositional Satisfiability

John Franco
franco@gauss.ececs.uc.edu

presentation to

SMT/SAT Summer School, Helsinki, 2013

July, 2013

Outline

Objective:

- Introduce SAT and SMT
- Show SAT and SMT algorithmic operations
- Show some complexity results

Discussion:

- Review various representations of Boolean functions
- Review some important operations within those representations
- Some algorithms have exponential time on some classes
- Modifications that may lead to polynomial time solutions
- Compare different operations
- Bounds on complexity
- Probabilistic analysis to show how algorithms get "stuck,"
  identify hard instances, measure "size" of easy classes,
  and suggest new algorithms that may be counter-intuitive

Terms and Connectives

A **variable** looks like this: v_9, takes a value from $\{0, 1\}$

A **positive literal**: v_9, a **negative literal**: $\overline{v_9}$

Boolean OR: $v_x \vee v_y = 1$ iff $v_x = 1$ or $v_y = 1$

Boolean AND: $v_x \wedge v_y = 1$ iff both $v_x = 1$ and $v_y = 1$

Boolean XOR: $v_x \oplus v_y = 1$ iff one of v_x, v_y has value 1

A **clause** looks like this: $(\overline{v_1} \vee \overline{v_2} \vee v_5 \vee v_9)$

An **instance** of SAT looks like this (CNF):

$$(v_1 \vee \overline{v_2} \vee v_7) \wedge (\overline{v_2} \vee v_6) \wedge (\overline{v_2} \vee \overline{v_4} \vee \overline{v_5}) \wedge (v_{10})...$$

Clause **width**: # literals; **k-SAT**: fixed width k

A clause of width 0 is empty and denoted $\emptyset$

Symbols ϕ and ψ denote an instance of SAT

An assignment of values satisfying ψ is a **model** for ψ

A Boolean Function

v_0	v_1	v_2	v_3	v_4	f
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	0	1	1
1	0	0	1	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	1
1	0	1	1	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	0	1	1
1	1	0	1	0	1
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	1
1	1	1	1	0	0
1	1	1	1	1	0

mapping of $\{0,1\}^n$ to $\{0,1\}$

Shown to the left is a truth table

Values in a row to the left of $|$ are an input assignment

The value to the right of $|$ in a row is the value of the function for that input assignment

Shannon's Expansion

At the dawn of the information age:

$$f(v_1, v_2, \ldots, v_n) = (v_1 \wedge f(1, v_2, \ldots, v_n)) \vee (\overline{v_1} \wedge f(0, v_2, \ldots, v_n))$$

$$f(v_1, v_2, \ldots, v_n) = (\overline{v_1} \vee f(1, v_2, \ldots, v_n)) \wedge (v_1 \vee f(0, v_2, \ldots, v_n))$$

Shannon's Expansion

At the dawn of the information age:

$$f(v_1, v_2, \ldots, v_n) = (v_1 \wedge f(1, v_2, \ldots, v_n)) \vee (\overline{v_1} \wedge f(0, v_2, \ldots, v_n))$$

$$f(v_1, v_2, \ldots, v_n) = (\overline{v_1} \vee f(1, v_2, \ldots, v_n)) \wedge (v_1 \vee f(0, v_2, \ldots, v_n))$$

normal forms:

$$\begin{aligned} f(v_1, v_2, ..., v_n) = \quad & (f(0,0,...,0) \wedge \overline{v_1} \wedge \overline{v_2} \wedge ... \wedge \overline{v_n}) \vee \\ & (f(1,0,...,0) \wedge v_1 \wedge \overline{v_2} \wedge ... \wedge \overline{v_n}) \vee \\ & (f(0,1,...,0) \wedge \overline{v_1} \wedge v_2 \wedge ... \wedge \overline{v_n}) \vee \\ & \quad ... \\ & (f(1,1,...,1) \wedge v_1 \wedge v_2 \wedge ... \wedge v_n) \end{aligned}$$

DNF

$$\begin{aligned} f(v_1, v_2, ..., v_n) = \quad & (f(0,0,...,0) \vee v_1 \vee v_2 \vee \ldots \vee v_n) \wedge \\ & (f(1,0,...,0) \vee \overline{v_1} \vee v_2 \vee \ldots \vee v_n) \wedge \\ & \quad ... \\ & (f(1,1,...,1) \vee \overline{v_1} \vee \overline{v_2} \vee \ldots \vee \overline{v_n}) \end{aligned}$$

CNF

CNF and DNF Representations

v_0	v_1	v_2	v_3	v_4	f
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	0	1	1
1	0	0	1	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	1
1	0	1	1	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	0	1	1
1	1	0	1	0	1
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	1
1	1	1	1	0	0
1	1	1	1	1	0

DNF:

$$(\overline{v_0} \wedge \overline{v_1} \wedge \overline{v_2} \wedge \overline{v_3} \wedge v_4) \vee$$
$$(\overline{v_0} \wedge \overline{v_1} \wedge \overline{v_2} \wedge v_3 \wedge v_4) \vee$$
$$(\overline{v_0} \wedge \overline{v_1} \wedge v_2 \wedge \overline{v_3} \wedge v_4) \vee$$
$$(\overline{v_0} \wedge \overline{v_1} \wedge v_2 \wedge v_3 \wedge v_4) \vee$$
$$\ldots$$
$$(v_0 \wedge v_1 \wedge v_2 \wedge \overline{v_3} \wedge v_4)$$

CNF:

$$(v_0 \vee v_1 \vee v_2 \vee v_3 \vee v_4) \wedge$$
$$(v_0 \vee v_1 \vee v_2 \vee \overline{v_3} \vee v_4) \wedge$$
$$(v_0 \vee v_1 \vee \overline{v_2} \vee v_3 \vee v_4) \wedge$$
$$(v_0 \vee v_1 \vee \overline{v_2} \vee \overline{v_3} \vee v_4) \wedge$$
$$\ldots$$
$$(\overline{v_0} \vee \overline{v_1} \vee \overline{v_2} \vee \overline{v_3} \vee \overline{v_4})$$

Resolution/Consensus

Resolution (CNF):

$$(\overline{v_1} \vee \overline{v_2} \vee v_i) \qquad (\overline{v_1} \vee v_3 \vee \overline{v_i})$$

$$(\overline{v_1} \vee \overline{v_2} \vee v_3)$$

If resolvent $= \emptyset$:
formula is unsatisfiable.
If cannot generate $\emptyset$:
formula is satisfiable.

Consensus (DNF):

$$(\overline{v_1} \wedge \overline{v_2} \wedge v_i) \qquad (\overline{v_1} \wedge v_3 \wedge \overline{v_i})$$

$$(\overline{v_1} \wedge \overline{v_2} \wedge v_3)$$

DNF is not minimized until consensus is no longer possible

Subsumption:

DNF: $(v_1 \wedge \overline{v_2})$ subsumes $(v_1 \wedge \overline{v_2} \wedge v_3 \wedge \overline{v_4})$

CNF: $(v_1 \vee \overline{v_2})$ subsumes $(v_1 \vee \overline{v_2} \vee v_3 \vee \overline{v_4})$

Reduced CNF and DNF Representations

v_0	v_1	v_2	v_3	v_4	f
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	0	1	1
1	0	0	1	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	1
1	0	1	1	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	0	1	1
1	1	0	1	0	1
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	1
1	1	1	1	0	0
1	1	1	1	1	0

DNF:

$$(v_1 \wedge \overline{v_2}) \vee$$
$$(\overline{v_3} \wedge v_4) \vee$$
$$(v_0 \wedge \overline{v_2}) \vee$$
$$(\overline{v_2} \wedge v_4)$$

CNF:

$$(v_0 \vee v_1 \vee v_4) \wedge$$
$$(\overline{v_0} \vee \overline{v_2} \vee \overline{v_3}) \wedge$$
$$(\overline{v_1} \vee \overline{v_2} \vee \overline{v_3}) \wedge$$
$$(\overline{v_2} \vee v_4)$$

Linear Time Translation to CNF

$\mathcal{O}_x$	Equivalent CNF Expression	Comment
0000	$(\overline{v_x})$	$v_x \Leftrightarrow 0$
1111	(v_x)	$v_x \Leftrightarrow 1$
0011	$(v_l \vee \overline{v_x}) \wedge (\overline{v_l} \vee v_x)$	
1100	$(v_l \vee v_x) \wedge (\overline{v_l} \vee \overline{v_x})$	$v_x \Leftrightarrow (\overline{v_l})$
0101	$(v_r \vee \overline{v_x}) \wedge (\overline{v_r} \vee v_x)$	
1010	$(v_r \vee v_x) \wedge (\overline{v_r} \vee \overline{v_x})$	$v_x \Leftrightarrow (\overline{v_r})$
0001	$(v_l \vee \overline{v_x}) \wedge (v_r \vee \overline{v_x}) \wedge (\overline{v_l} \vee \overline{v_r} \vee v_x)$	$v_x \Leftrightarrow (v_l \wedge v_r)$
1110	$(v_l \vee v_x) \wedge (v_r \vee v_x) \wedge (\overline{v_l} \vee \overline{v_r} \vee \overline{v_x})$	
0010	$(\overline{v_l} \vee \overline{v_x}) \wedge (v_r \vee \overline{v_x}) \wedge (v_l \vee \overline{v_r} \vee v_x)$	
1101	$(v_l \vee v_x) \wedge (\overline{v_r} \vee v_x) \wedge (\overline{v_l} \vee v_r \vee \overline{v_x})$	$v_x \Leftrightarrow (v_l \rightarrow v_r)$
0100	$(v_l \vee \overline{v_x}) \wedge (\overline{v_r} \vee \overline{v_x}) \wedge (\overline{v_l} \vee v_r \vee v_x)$	
1011	$(\overline{v_l} \vee v_x) \wedge (v_r \vee v_x) \wedge (v_l \vee \overline{v_r} \vee \overline{v_x})$	$v_x \Leftrightarrow (v_l \leftarrow v_r)$
1000	$(\overline{v_l} \vee \overline{v_x}) \wedge (\overline{v_r} \vee \overline{v_x}) \wedge (v_l \vee v_r \vee v_x)$	
0111	$(\overline{v_l} \vee v_x) \wedge (\overline{v_r} \vee v_x) \wedge (v_l \vee v_r \vee \overline{v_x})$	$v_x \Leftrightarrow (v_l \vee v_r)$
1001	$(v_l \vee \overline{v_r} \vee \overline{v_x}) \wedge (\overline{v_l} \vee v_r \vee \overline{v_x}) \wedge (\overline{v_l} \vee \overline{v_r} \vee v_x) \wedge (v_l \vee v_r \vee v_x)$	$v_x \Leftrightarrow (v_l \Leftrightarrow v_r)$
0110	$(\overline{v_l} \vee v_r \vee v_x) \wedge (v_l \vee \overline{v_r} \vee v_x) \wedge (v_l \vee v_r \vee \overline{v_x}) \wedge (\overline{v_l} \vee \overline{v_r} \vee \overline{v_x})$	$v_x \Leftrightarrow (v_l \oplus v_r)$

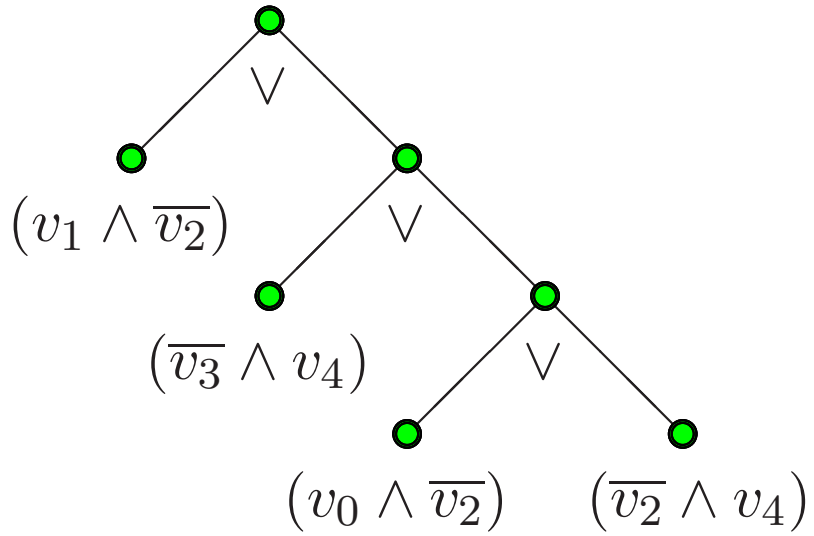
CNF expressions equivalent to $v_x \Leftrightarrow (v_l \ \mathcal{O}_x \ v_r)$ for $\mathcal{O}_x$ as shown

Linear Time Translation to CNF

Example:

$$(v_1 \wedge \overline{v_2}) \vee (\overline{v_3} \vee v_4) \vee (v_0 \wedge \overline{v_2}) \vee (\overline{v_2} \wedge v_4)$$

Translation: (to obtain equi-satisfiable CNF formula)

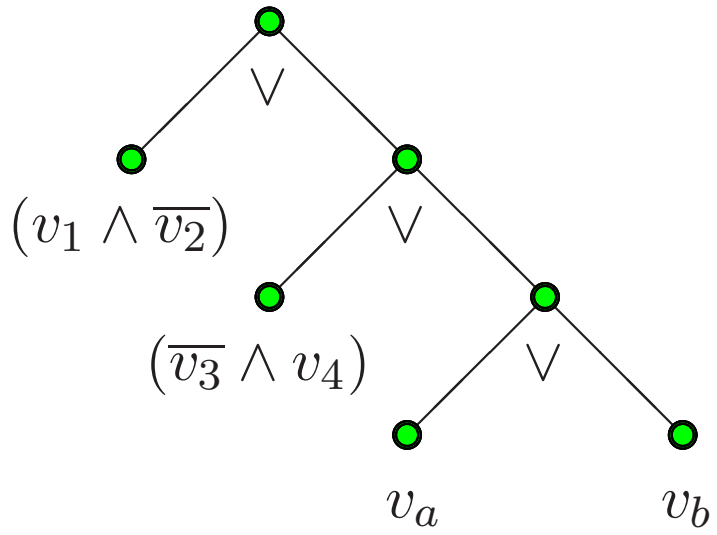


Linear Time Translation to CNF

Example:

$$(v_1 \wedge \overline{v_2}) \vee (\overline{v_3} \vee v_4) \vee (v_0 \wedge \overline{v_2}) \vee (\overline{v_2} \wedge v_4)$$

Translation: (to obtain equi-satisfiable CNF formula)



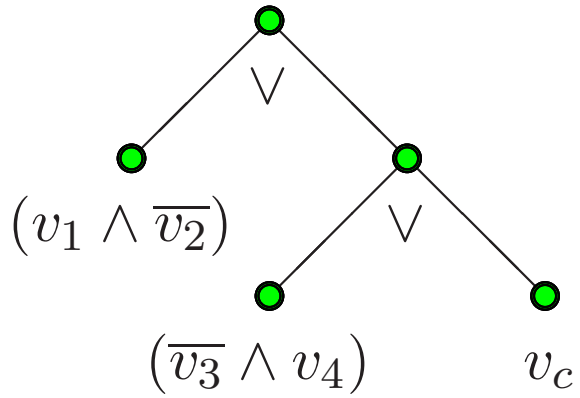
$$(v_0 \vee \overline{v_a}) \wedge (\overline{v_2} \vee \overline{v_a}) \wedge (\overline{v_0} \vee v_2 \vee v_a) \wedge (\overline{v_2} \vee \overline{v_b}) \wedge (v_4 \vee \overline{v_b}) \wedge (v_2 \vee \overline{v_4} \vee v_b)$$

Linear Time Translation to CNF

Example:

$$(v_1 \wedge \overline{v_2}) \vee (\overline{v_3} \vee v_4) \vee (v_0 \wedge \overline{v_2}) \vee (\overline{v_2} \wedge v_4)$$

Translation: (to obtain equi-satisfiable CNF formula)



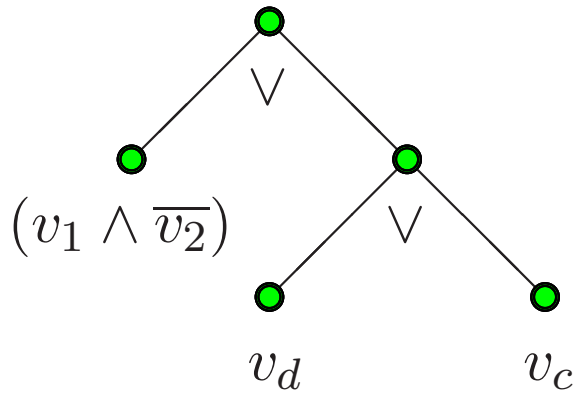
$$(v_0 \vee \overline{v_a}) \wedge (\overline{v_2} \vee \overline{v_a}) \wedge (\overline{v_0} \vee v_2 \vee v_a) \wedge (\overline{v_2} \vee \overline{v_b}) \wedge (v_4 \vee \overline{v_b}) \wedge (v_2 \vee \overline{v_4} \vee v_b) \wedge$$
$$(\overline{v_a} \vee v_c) \wedge (\overline{v_b} \vee v_c) \wedge (v_a \vee v_b \vee \overline{v_c})$$

Linear Time Translation to CNF

Example:

$$(v_1 \wedge \overline{v_2}) \vee (\overline{v_3} \vee v_4) \vee (v_0 \wedge \overline{v_2}) \vee (\overline{v_2} \wedge v_4)$$

Translation: (to obtain equi-satisfiable CNF formula)



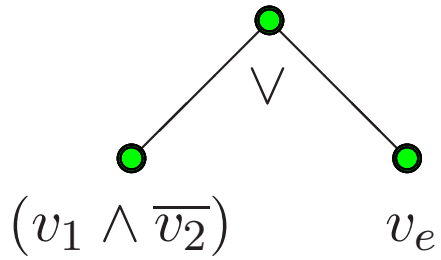
$$(v_0 \vee \overline{v_a}) \wedge (\overline{v_2} \vee \overline{v_a}) \wedge (\overline{v_0} \vee v_2 \vee v_a) \wedge (\overline{v_2} \vee \overline{v_b}) \wedge (v_4 \vee \overline{v_b}) \wedge (v_2 \vee \overline{v_4} \vee v_b) \wedge$$
$$(\overline{v_a} \vee v_c) \wedge (\overline{v_b} \vee v_c) \wedge (v_a \vee v_b \vee \overline{v_c}) \wedge (\overline{v_3} \vee \overline{v_d}) \wedge (v_4 \vee \overline{v_d}) \wedge (\overline{v_3} \vee v_4 \vee v_d)$$

Linear Time Translation to CNF

Example:

$$(v_1 \wedge \overline{v_2}) \vee (\overline{v_3} \vee v_4) \vee (v_0 \wedge \overline{v_2}) \vee (\overline{v_2} \wedge v_4)$$

Translation: (to obtain equi-satisfiable CNF formula)



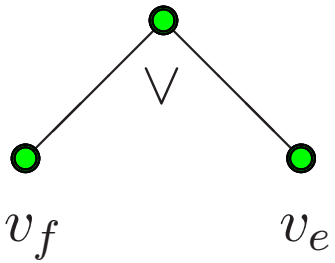
$$(v_0 \vee \overline{v_a}) \wedge (\overline{v_2} \vee \overline{v_a}) \wedge (\overline{v_0} \vee v_2 \vee v_a) \wedge (\overline{v_2} \vee \overline{v_b}) \wedge (v_4 \vee \overline{v_b}) \wedge (v_2 \vee \overline{v_4} \vee v_b) \wedge$$
$$(\overline{v_a} \vee v_c) \wedge (\overline{v_b} \vee v_c) \wedge (v_a \vee v_b \vee \overline{v_c}) \wedge (\overline{v_3} \vee \overline{v_d}) \wedge (v_4 \vee \overline{v_d}) \wedge (\overline{v_3} \vee v_4 \vee v_d) \wedge$$
$$(\overline{v_d} \vee v_e) \wedge (\overline{v_c} \vee v_e) \wedge (v_d \vee v_c \vee \overline{v_e})$$

Linear Time Translation to CNF

Example:

$$(v_1 \wedge \overline{v_2}) \vee (\overline{v_3} \vee v_4) \vee (v_0 \wedge \overline{v_2}) \vee (\overline{v_2} \wedge v_4)$$

Translation: (to obtain equi-satisfiable CNF formula)



$$(v_0 \vee \overline{v_a}) \wedge (\overline{v_2} \vee \overline{v_a}) \wedge (\overline{v_0} \vee v_2 \vee v_a) \wedge (\overline{v_2} \vee \overline{v_b}) \wedge (v_4 \vee \overline{v_b}) \wedge (v_2 \vee \overline{v_4} \vee v_b) \wedge$$
$$(\overline{v_a} \vee v_c) \wedge (\overline{v_b} \vee v_c) \wedge (v_a \vee v_b \vee \overline{v_c}) \wedge (\overline{v_3} \vee \overline{v_d}) \wedge (v_4 \vee \overline{v_d}) \wedge (\overline{v_3} \vee v_4 \vee v_d) \wedge$$
$$(\overline{v_d} \vee v_e) \wedge (\overline{v_c} \vee v_e) \wedge (v_d \vee v_c \vee \overline{v_e}) \wedge (v_1 \vee \overline{v_f}) \wedge (\overline{v_2} \vee \overline{v_f}) \wedge (\overline{v_1} \vee v_2 \vee v_f)$$

Linear Time Translation to CNF

Example:

$$(v_1 \wedge \overline{v_2}) \vee (\overline{v_3} \vee v_4) \vee (v_0 \wedge \overline{v_2}) \vee (\overline{v_2} \wedge v_4)$$

Translation: (to obtain equi-satisfiable CNF formula)

v_g

$$(v_0 \vee \overline{v_a}) \wedge (\overline{v_2} \vee \overline{v_a}) \wedge (\overline{v_0} \vee v_2 \vee v_a) \wedge (\overline{v_2} \vee \overline{v_b}) \wedge (v_4 \vee \overline{v_b}) \wedge (v_2 \vee \overline{v_4} \vee v_b) \wedge$$
$$(\overline{v_a} \vee v_c) \wedge (\overline{v_b} \vee v_c) \wedge (v_a \vee v_b \vee \overline{v_c}) \wedge (\overline{v_3} \vee \overline{v_d}) \wedge (v_4 \vee \overline{v_d}) \wedge (\overline{v_3} \vee v_4 \vee v_d) \wedge$$
$$(\overline{v_d} \vee v_e) \wedge (\overline{v_c} \vee v_e) \wedge (v_d \vee v_c \vee \overline{v_e}) \wedge (v_1 \vee \overline{v_f}) \wedge (\overline{v_2} \vee \overline{v_f}) \wedge (\overline{v_1} \vee v_2 \vee v_f) \wedge$$
$$(\overline{v_f} \vee v_g) \wedge (\overline{v_e} \vee v_g) \wedge (v_f \vee v_e \vee \overline{v_g})$$

Linear Time Translation to CNF

Example:

$$(v_1 \wedge \overline{v_2}) \vee (\overline{v_3} \vee v_4) \vee (v_0 \wedge \overline{v_2}) \vee (\overline{v_2} \wedge v_4)$$

Translation: (to obtain equi-satisfiable CNF formula)

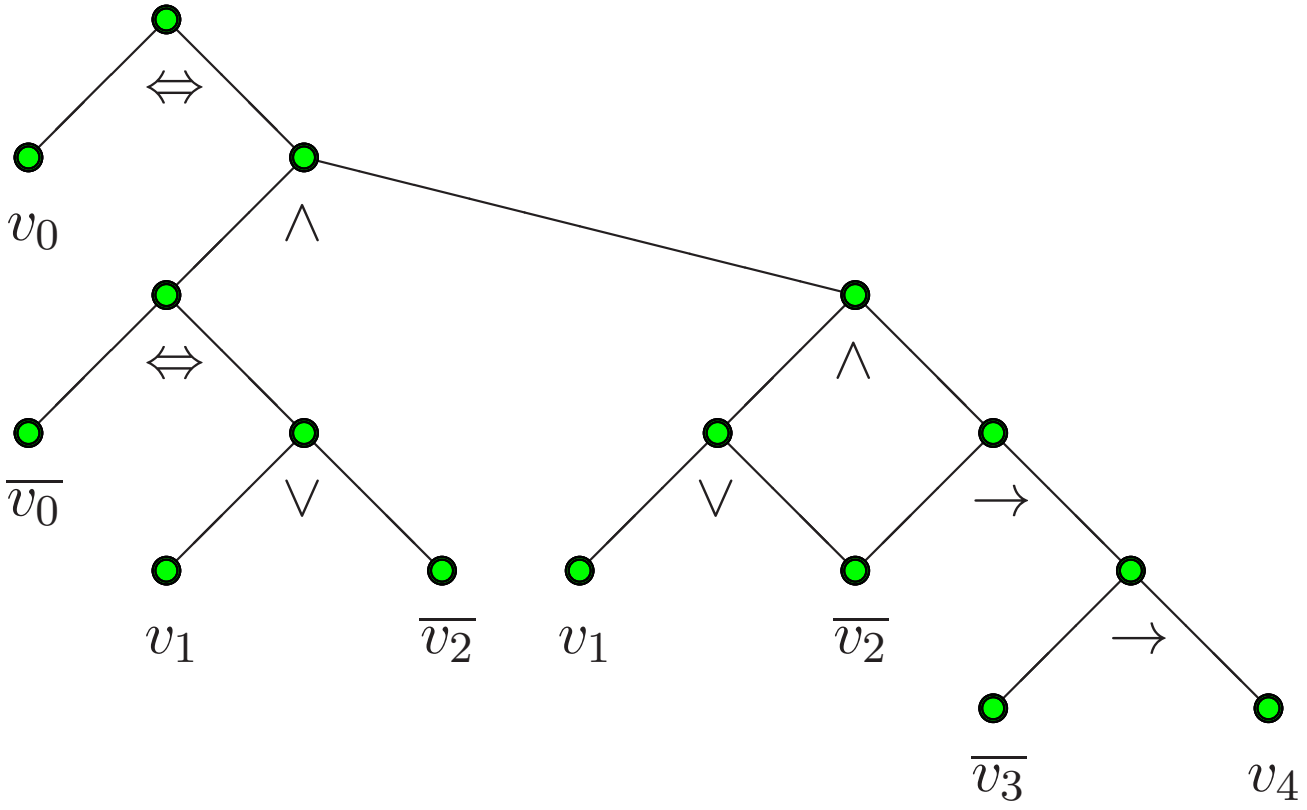
$$(v_0 \vee \overline{v_a}) \wedge (\overline{v_2} \vee \overline{v_a}) \wedge (\overline{v_0} \vee v_2 \vee v_a) \wedge (\overline{v_2} \vee \overline{v_b}) \wedge (v_4 \vee \overline{v_b}) \wedge (v_2 \vee \overline{v_4} \vee v_b) \wedge$$
$$(\overline{v_a} \vee v_c) \wedge (\overline{v_b} \vee v_c) \wedge (v_a \vee v_b \vee \overline{v_c}) \wedge (\overline{v_3} \vee \overline{v_d}) \wedge (v_4 \vee \overline{v_d}) \wedge (\overline{v_3} \vee v_4 \vee v_d) \wedge$$
$$(\overline{v_d} \vee v_e) \wedge (\overline{v_c} \vee v_e) \wedge (v_d \vee v_c \vee \overline{v_e}) \wedge (v_1 \vee \overline{v_f}) \wedge (\overline{v_2} \vee \overline{v_f}) \wedge (\overline{v_1} \vee v_2 \vee v_f) \wedge$$
$$(\overline{v_f} \vee v_g) \wedge (\overline{v_e} \vee v_g) \wedge (v_f \vee v_e \vee \overline{v_g}) \wedge (v_g)$$

Linear Time Translation to CNF

Example:

$$v_0 \Leftrightarrow ((\overline{v_0} \Leftrightarrow (v_1 \vee \overline{v_2})) \wedge (v_1 \vee \overline{v_2}) \wedge (\overline{v_2} \rightarrow \overline{v_3} \rightarrow v_4))$$

Parse DAG:



Linear Time Translation to CNF

Example:

$$v_0 \Leftrightarrow ((\overline{v_0} \Leftrightarrow (v_1 \vee \overline{v_2})) \wedge (v_1 \vee \overline{v_2}) \wedge (\overline{v_2} \rightarrow \overline{v_3} \rightarrow v_4))$$

Translation: (to obtain equi-satisfiable CNF formula)

$$
\begin{array}{l}
(v_0 \vee v_{x_1}) \wedge (\overline{v_0} \vee \overline{v_{x_1}}) \wedge \\
(v_2 \vee v_{x_2}) \wedge (\overline{v_2} \vee \overline{v_{x_2}}) \wedge \\
(v_3 \vee v_{x_3}) \wedge (\overline{v_3} \vee \overline{v_{x_3}}) \wedge \\
(\overline{v_1} \vee v_{x_4}) \wedge (\overline{v_{x_2}} \vee v_{x_4}) \wedge (v_1 \vee v_{x_2} \vee \overline{v_{x_4}}) \wedge \\
(v_{x_3} \vee v_{x_5}) \wedge (\overline{v_4} \vee v_{x_5}) \wedge (\overline{v_{x_3}} \vee v_4 \vee \overline{v_{x_5}}) \wedge \\
(v_{x_1} \vee \overline{v_{x_4}} \vee \overline{v_{x_6}}) \wedge (\overline{v_{x_1}} \vee v_{x_4} \vee \overline{v_{x_6}}) \wedge \\
\quad (\overline{v_{x_1}} \vee \overline{v_{x_4}} \vee v_{x_6}) \wedge (v_{x_1} \vee v_{x_4} \vee v_{x_6}) \wedge \\
(v_{x_2} \vee v_{x_7}) \wedge (\overline{v_{x_5}} \vee v_{x_7}) \wedge (\overline{v_{x_2}} \vee v_{x_5} \vee \overline{v_{x_7}}) \wedge \\
(v_{x_4} \vee \overline{v_{x_8}}) \wedge (v_{x_7} \vee \overline{v_{x_8}}) \wedge (\overline{v_{x_4}} \vee \overline{v_{x_7}} \vee v_{x_8}) \wedge \\
(v_{x_6} \vee \overline{v_{x_9}}) \wedge (v_{x_8} \vee \overline{v_{x_9}}) \wedge (\overline{v_{x_6}} \vee \overline{v_{x_8}} \vee v_{x_9}) \wedge \\
(v_0 \vee \overline{v_{x_9}} \vee \overline{v_{x_{10}}}) \wedge (\overline{v_0} \vee v_{x_9} \vee \overline{v_{x_{10}}}) \wedge \\
\quad (\overline{v_0} \vee \overline{v_{x_9}} \vee v_{x_{10}}) \wedge (v_0 \vee v_{x_9} \vee v_{x_{10}}) \wedge \\
(v_{x_{10}})
\end{array}
$$

Regular Resolution

Davis-Putnam (CNF):

$$(\overline{v_1} \vee \overline{v_2} \vee v_i) \wedge (\overline{v_i} \vee v_3) \wedge (\overline{v_2} \vee v_4) \wedge (v_1 \vee \overline{v_i}) \wedge (v_i \vee v_3)$$

collect all resolvents with v_i:

$$(\overline{v_1} \vee \overline{v_2} \vee v_i) \wedge (\overline{v_i} \vee v_3) \wedge (\overline{v_2} \vee v_4) \wedge (v_1 \vee \overline{v_i}) \wedge (v_i \vee v_3) \wedge$$
$$(\overline{v_1} \vee \overline{v_2} \vee v_3) \wedge (v_3) \wedge (v_1 \vee v_3)$$

remove all clauses with v_i:

$$(\overline{v_2} \vee v_4) \wedge (\overline{v_1} \vee \overline{v_2} \vee v_3) \wedge (v_3) \wedge (v_1 \vee v_3)$$

Regular Resolution

Davis-Putnam (CNF):

$$(\overline{v_1} \vee \overline{v_2} \vee v_i) \wedge (\overline{v_i} \vee v_3) \wedge (\overline{v_2} \vee v_4) \wedge (v_1 \vee \overline{v_i}) \wedge (v_i \vee v_3)$$

collect all resolvents with v_i:

$$(\overline{v_1} \vee \overline{v_2} \vee v_i) \wedge (\overline{v_i} \vee v_3) \wedge (\overline{v_2} \vee v_4) \wedge (v_1 \vee \overline{v_i}) \wedge (v_i \vee v_3) \wedge$$
$$(\overline{v_1} \vee \overline{v_2} \vee v_3) \wedge (v_3) \wedge (v_1 \vee v_3)$$

remove all clauses with v_i:

$$(\overline{v_2} \vee v_4) \wedge (\overline{v_1} \vee \overline{v_2} \vee v_3) \wedge (v_3) \wedge (v_1 \vee v_3)$$

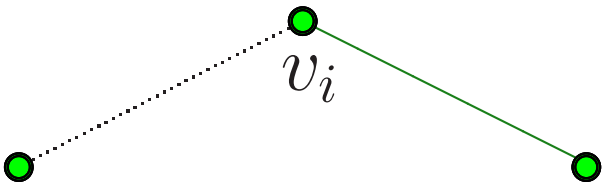
Do this repeatedly
If an empty resolvent ($\emptyset$) is generated, the instance has no model
Otherwise, there is a model

A Search Procedure

Davis-Putnam-Loveland-Logemann (CNF):

splitting rule:

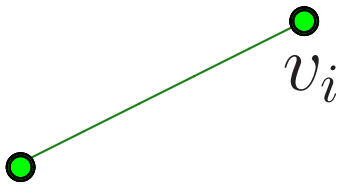
$$(\overline{v_1} \vee \overline{v_2} \vee v_i) \wedge (\overline{v_i} \vee v_3) \wedge (\overline{v_2} \vee v_4)$$



$$(\overline{v_1} \vee \overline{v_2}) \wedge (\overline{v_2} \vee v_4) \qquad (v_3) \wedge (\overline{v_2} \vee v_4)$$

unit clause rule:

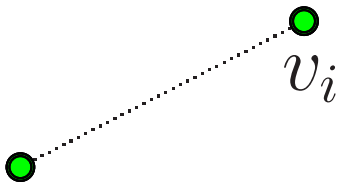
$$(\overline{v_1} \vee \overline{v_2} \vee \overline{v_i}) \wedge (v_i)..$$



$$(\overline{v_1} \vee \overline{v_2})..$$

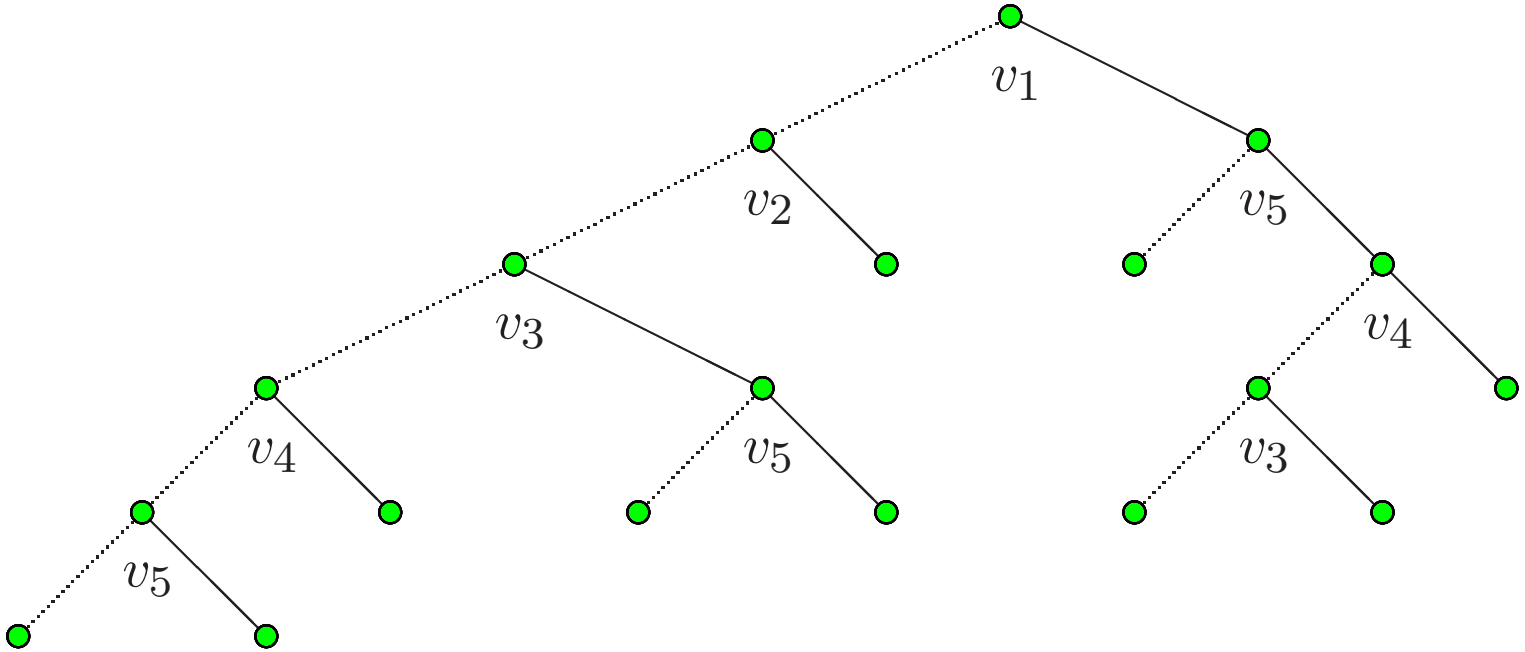
pure literal rule:

$$(\overline{v_1} \vee \overline{v_i}) \wedge (v_2 \vee \overline{v_i}) \wedge (v_3 \vee \overline{v_4})..$$



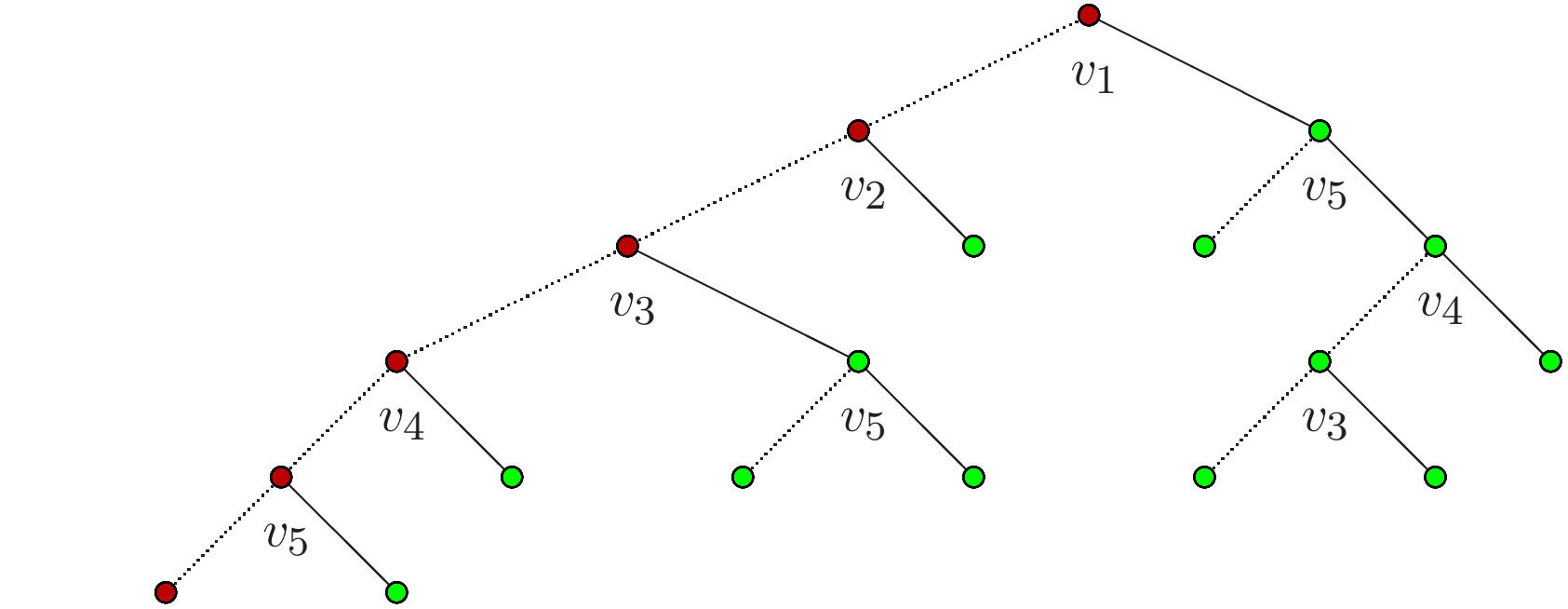
$$(v_3 \vee \overline{v_4})..$$

DPLL Refutation



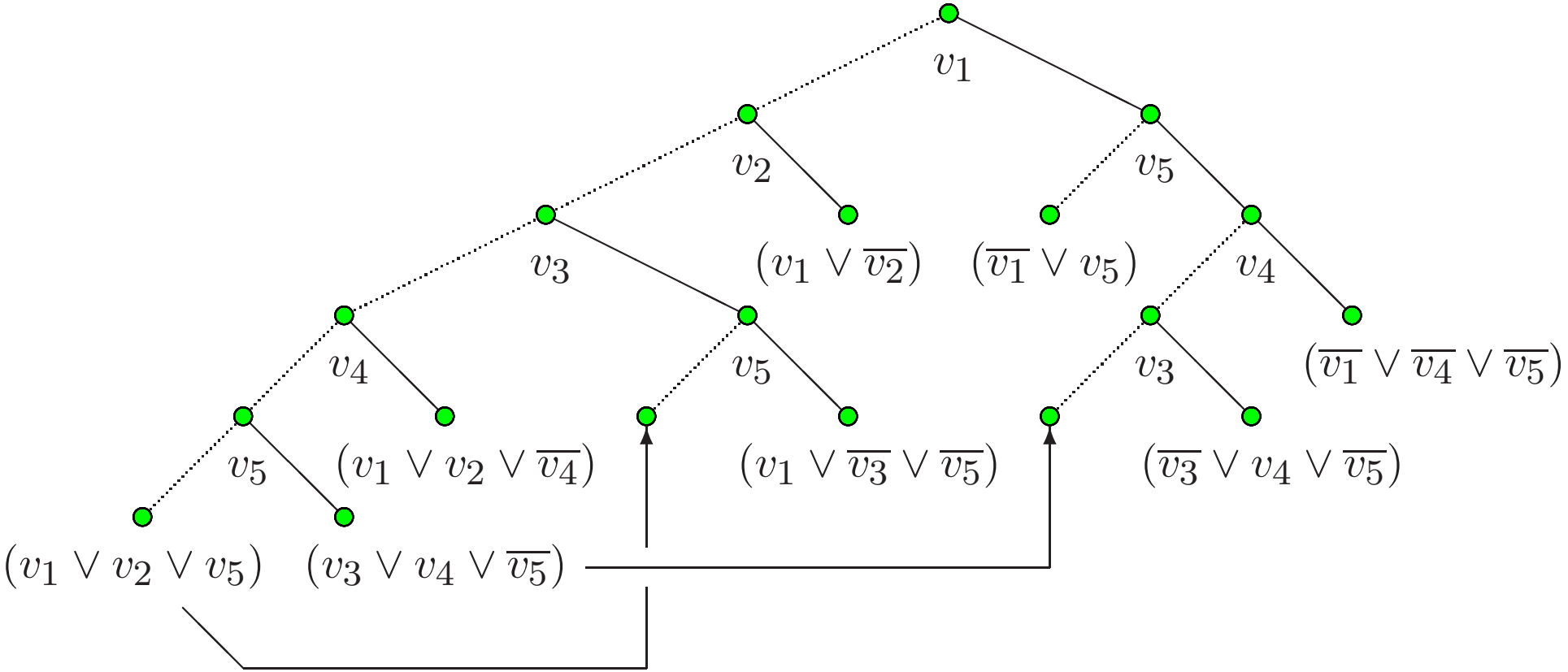
$$(v_1 \vee v_2 \vee v_5) \wedge (v_3 \vee v_4 \vee \overline{v_5}) \wedge (v_1 \vee v_2 \vee \overline{v_4}) \wedge (v_1 \vee \overline{v_3} \vee \overline{v_5}) \wedge$$
$$(v_1 \vee \overline{v_2}) \wedge (\overline{v_3} \vee v_4 \vee \overline{v_5}) \wedge (\overline{v_1} \vee \overline{v_4} \vee \overline{v_5}) \wedge (\overline{v_1} \vee v_5)$$

DPLL Refutation



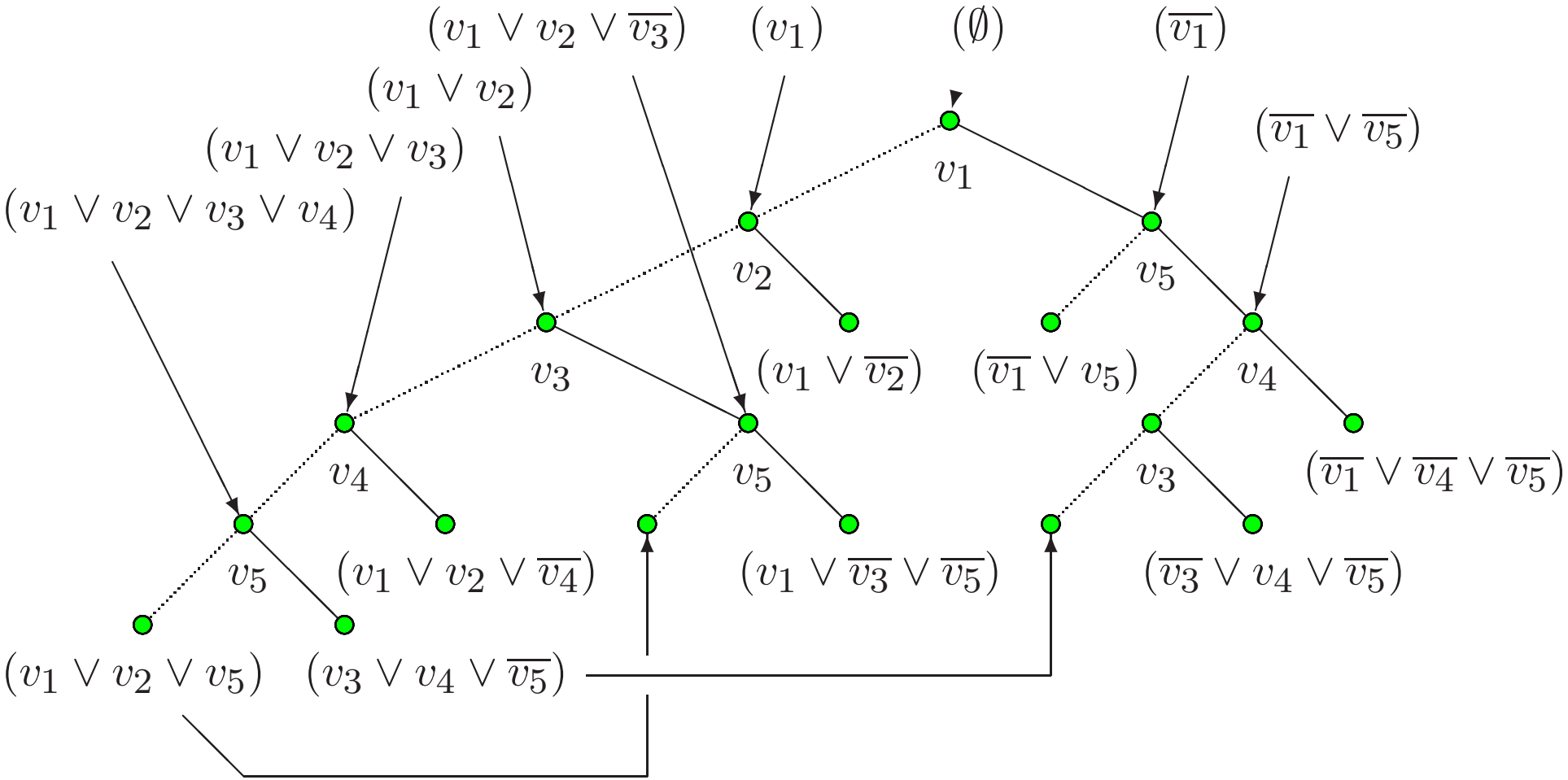
$$(v_1 \vee v_2 \vee v_5) \wedge (v_3 \vee v_4 \vee \overline{v_5}) \wedge (v_1 \vee v_2 \vee \overline{v_4}) \wedge (v_1 \vee \overline{v_3} \vee \overline{v_5}) \wedge$$
$$(v_1 \vee \overline{v_2}) \wedge (\overline{v_3} \vee v_4 \vee \overline{v_5}) \wedge (\overline{v_1} \vee \overline{v_4} \vee \overline{v_5}) \wedge (\overline{v_1} \vee v_5)$$

DPLL is Resolution



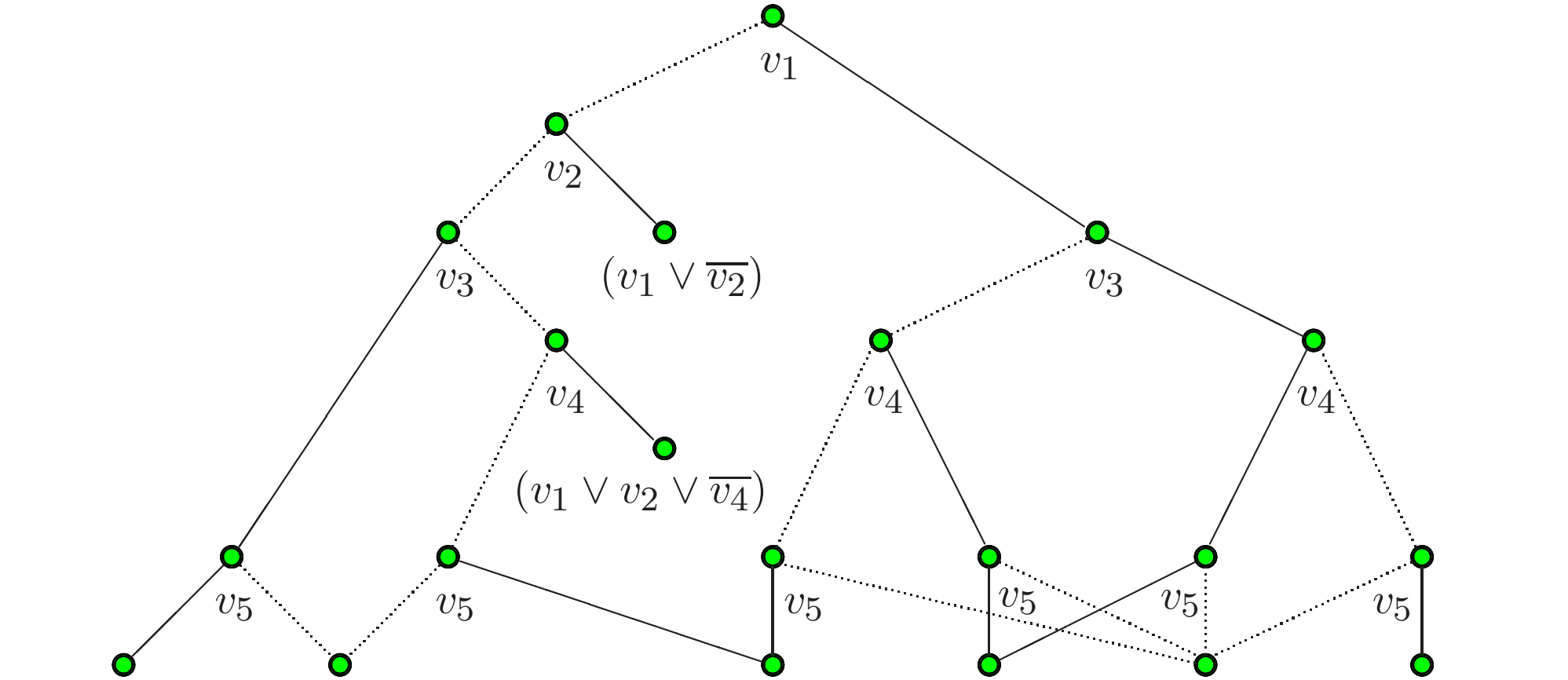
$$(v_1 \vee v_2 \vee v_5) \wedge (v_3 \vee v_4 \vee \overline{v_5}) \wedge (v_1 \vee v_2 \vee \overline{v_4}) \wedge (v_1 \vee \overline{v_3} \vee \overline{v_5}) \wedge$$
$$(v_1 \vee \overline{v_2}) \wedge (\overline{v_3} \vee v_4 \vee \overline{v_5}) \wedge (\overline{v_1} \vee \overline{v_4} \vee \overline{v_5}) \wedge (\overline{v_1} \vee v_5)$$

DPLL is Resolution



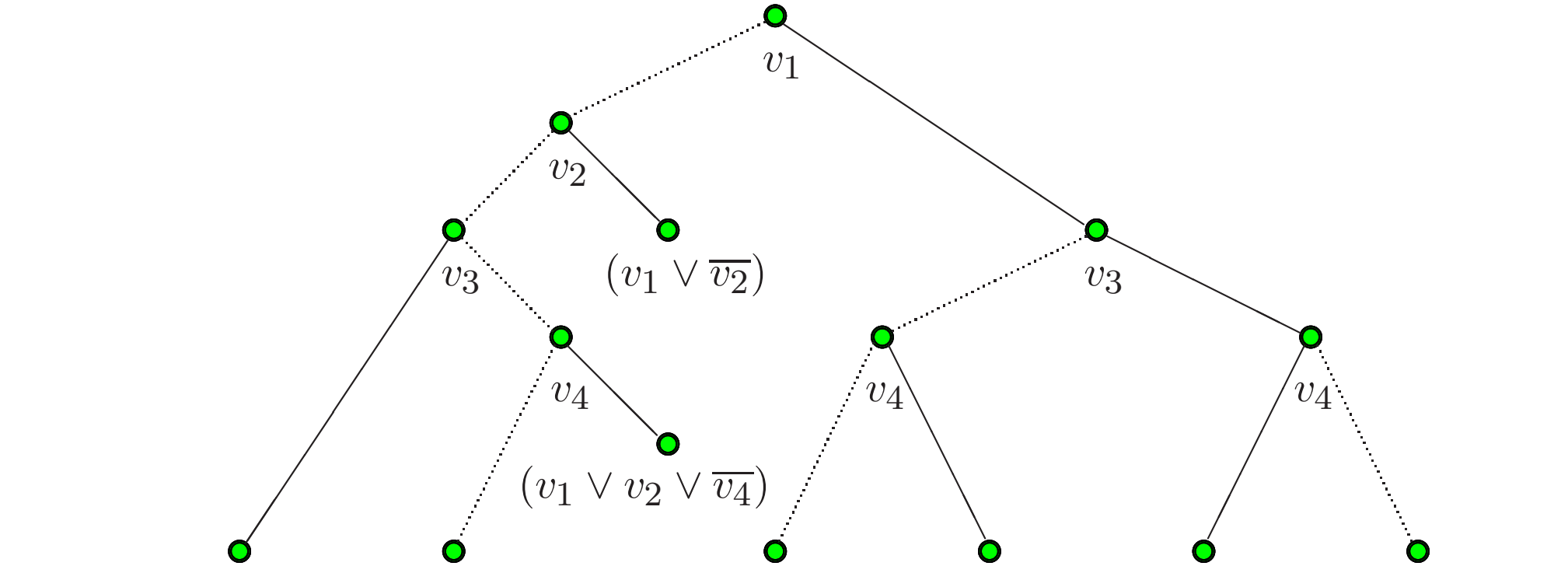
$$(v_1 \vee v_2 \vee v_5) \wedge (v_3 \vee v_4 \vee \overline{v_5}) \wedge (v_1 \vee v_2 \vee \overline{v_4}) \wedge (v_1 \vee \overline{v_3} \vee \overline{v_5}) \wedge$$
$$(v_1 \vee \overline{v_2}) \wedge (\overline{v_3} \vee v_4 \vee \overline{v_5}) \wedge (\overline{v_1} \vee \overline{v_4} \vee \overline{v_5}) \wedge (\overline{v_1} \vee v_5)$$

DP in DPLL



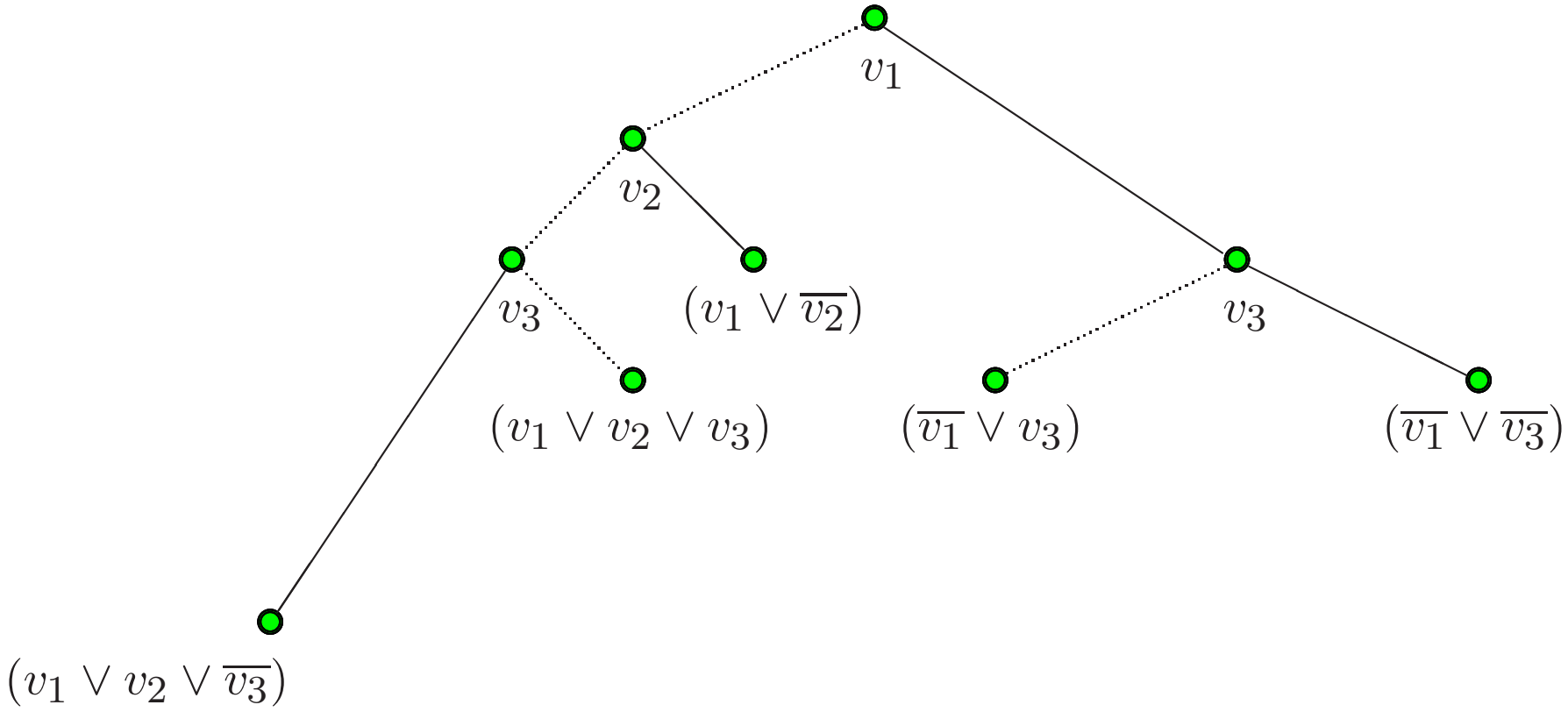
$$(v_1 \vee v_2 \vee v_5) \wedge (v_3 \vee v_4 \vee \overline{v_5}) \wedge (v_1 \vee v_2 \vee \overline{v_4}) \wedge (v_1 \vee \overline{v_3} \vee \overline{v_5}) \wedge$$
$$(v_1 \vee \overline{v_2}) \wedge (\overline{v_3} \vee v_4 \vee \overline{v_5}) \wedge (\overline{v_1} \vee \overline{v_4} \vee \overline{v_5}) \wedge (\overline{v_1} \vee v_5)$$

DP in DPLL



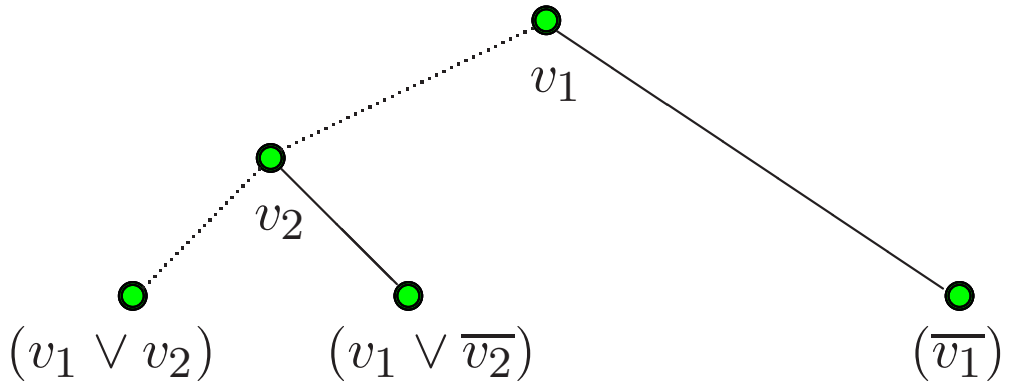
$$(v_1 \vee v_2 \vee v_5) \wedge (v_3 \vee v_4 \vee \overline{v_5}) \wedge (v_1 \vee v_2 \vee \overline{v_4}) \wedge (v_1 \vee \overline{v_3} \vee \overline{v_5}) \wedge$$
$$(v_1 \vee \overline{v_2}) \wedge (\overline{v_3} \vee v_4 \vee \overline{v_5}) \wedge (\overline{v_1} \vee \overline{v_4} \vee \overline{v_5}) \wedge (\overline{v_1} \vee v_5)$$

DP in DPLL



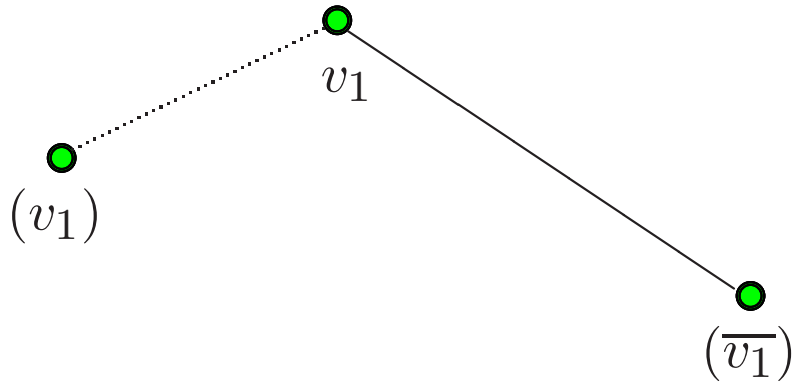
$$(v_1 \vee v_2 \vee v_5) \wedge (v_3 \vee v_4 \vee \overline{v_5}) \wedge (v_1 \vee v_2 \vee \overline{v_4}) \wedge (v_1 \vee \overline{v_3} \vee \overline{v_5}) \wedge$$
$$(v_1 \vee \overline{v_2}) \wedge (\overline{v_3} \vee v_4 \vee \overline{v_5}) \wedge (\overline{v_1} \vee \overline{v_4} \vee \overline{v_5}) \wedge (\overline{v_1} \vee v_5)$$

DP in DPLL



$$(v_1 \vee v_2 \vee v_5) \wedge (v_3 \vee v_4 \vee \overline{v_5}) \wedge (v_1 \vee v_2 \vee \overline{v_4}) \wedge (v_1 \vee \overline{v_3} \vee \overline{v_5}) \wedge$$
$$(v_1 \vee \overline{v_2}) \wedge (\overline{v_3} \vee v_4 \vee \overline{v_5}) \wedge (\overline{v_1} \vee \overline{v_4} \vee \overline{v_5}) \wedge (\overline{v_1} \vee v_5)$$

DP in DPLL



$$(v_1 \vee v_2 \vee v_5) \wedge (v_3 \vee v_4 \vee \overline{v_5}) \wedge (v_1 \vee v_2 \vee \overline{v_4}) \wedge (v_1 \vee \overline{v_3} \vee \overline{v_5}) \wedge$$
$$(v_1 \vee \overline{v_2}) \wedge (\overline{v_3} \vee v_4 \vee \overline{v_5}) \wedge (\overline{v_1} \vee \overline{v_4} \vee \overline{v_5}) \wedge (\overline{v_1} \vee v_5)$$

DP in DPLL

$(\emptyset)$

$$(v_1 \vee v_2 \vee v_5) \wedge (v_3 \vee v_4 \vee \overline{v_5}) \wedge (v_1 \vee v_2 \vee \overline{v_4}) \wedge (v_1 \vee \overline{v_3} \vee \overline{v_5}) \wedge$$
$$(v_1 \vee \overline{v_2}) \wedge (\overline{v_3} \vee v_4 \vee \overline{v_5}) \wedge (\overline{v_1} \vee \overline{v_4} \vee \overline{v_5}) \wedge (\overline{v_1} \vee v_5)$$

Resolution vs. DPLL Complexity

Minimum size of a refutation

If there is a DPLL search involving n choicepoints for instance `x`
then there's a resolution refutation of length $p(n)$ for instance `x`

By adding some components to DPLL, namely
clause learning and restarts, DPLL can p-simulate resolution

Refutation Size and Max Clause Width

Minimum size of a refutation

$$S(\psi) = e^{\Omega\left(\frac{(w(\psi \vdash \emptyset) - w(\psi))^2}{|V|}\right)}$$

where

ψ is a CNF formula with variable set V.

$w(\psi)$ is the width of the widest clause in ψ.

$w(\psi \vdash \emptyset)$ is the minimum of the width of the widest clause over all refutations of ψ.

$S(\psi)$ is the minimum size of a resolution refutation of ψ.

when

Sparseness: few pairs of clauses have a common literal or complementary pair of literals.

Upper Bounds

Autarky

A partial assignment that satisfies all those clauses affected by it.
examples: a pure literal; $v_1 = v_2 = 1$ below.

$$(\overline{v_1} \vee v_2 \vee v_3) \wedge (v_1 \vee \overline{v_2} \vee \overline{v_4}) \wedge (\overline{v_3} \vee v_4 \vee \overline{v_5}) \wedge (v_1 \vee v_4 \vee v_5)$$

A subformula obtained by applying an autarky is satisfiable
if and only if the formula is.

Algorithm Idea for k-CNF formulas - $2^{0.695n}$ for 3-SAT

Choose smallest clause $c = (l_{\pi_1} \vee \ldots \vee l_{\pi_i})$
Create i subproblems using the following assignments:

$l_{\pi_1} = 1$

$l_{\pi_1} = 0, l_{\pi_2} = 1$

$\ldots$

$l_{\pi_1} = 0, l_{\pi_2} = 0, \ldots, l_{\pi_{i-1}} = 0, l_{\pi_i} = 1$

If $\exists$ subformula with width k clauses, remove and continue
Otherwise solve subformulas

Upper Bounds

A probabilistic algorithm

Repeat the following $(2-2/k)^n$ times:
  Randomly choose an assignment M.
  If M is a model for ψ return M.
  Repeat the following $3n$ times:
    Pick an unsatisfied clause $c \in \psi$.
    Randomly choose $l \in c$.
    Reverse the value of the variable associated with l.
    If the updated M is a model for ψ return M.
Return "unsatisfiable?"

Probability a model is found in $3n$ steps is $p \geq (2/3)(2-2/k)^{-n}$

Hence the algorithm solves k-SAT in time $|\psi|^{O(1)}(2-2/k)^n$

($|\psi|^{O(1)}2^{0.42n}$ for $k=3$) with error probability $o(1)$

Look at the SAT Handbook for more information

Resolution Can Be Bad

Prove that it is impossible to assign $n+1$ pigeons to n holes without at least one hole containing two pigeons

Variables	Subscript Range	Meaning
$v_{i,k}$	$1 \leq i \leq n$ $1 \leq k \leq n+1$	$v_{i,k} = 1$ iff the k^{th} pigeon is in hole i

Clauses	Subscript Range	Meaning
$(v_{1,k} \vee \ldots \vee v_{n,k})$	$1 \leq k \leq n+1$	Every pigeon in at least one hole
$(\overline{v_{i,l}} \vee \overline{v_{i,k}})$	$1 \leq l < k \leq n+1$ $1 \leq i \leq n$	Each hole has at most one pigeon

Every resolution proof requires generating exponentially many resolvents

Extended Resolution

Just add this:

$$w \Leftrightarrow f(x, y, \ldots, z)$$

where w is a variable not already in the formula and f is any Boolean function of variables that are in the formula.

Example:

$$(w \vee x) \wedge (w \vee y) \wedge (\overline{w} \vee \overline{x} \vee \overline{y})$$

This is equivalent to:

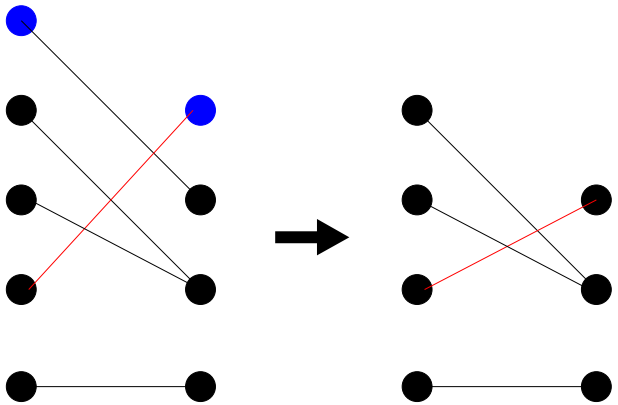
$$w \Leftrightarrow (\overline{x} \vee \overline{y})$$

which means either x and y both have value 1 (then $w = 0$) or at least one of x or y has value 0 (then $w = 1$).

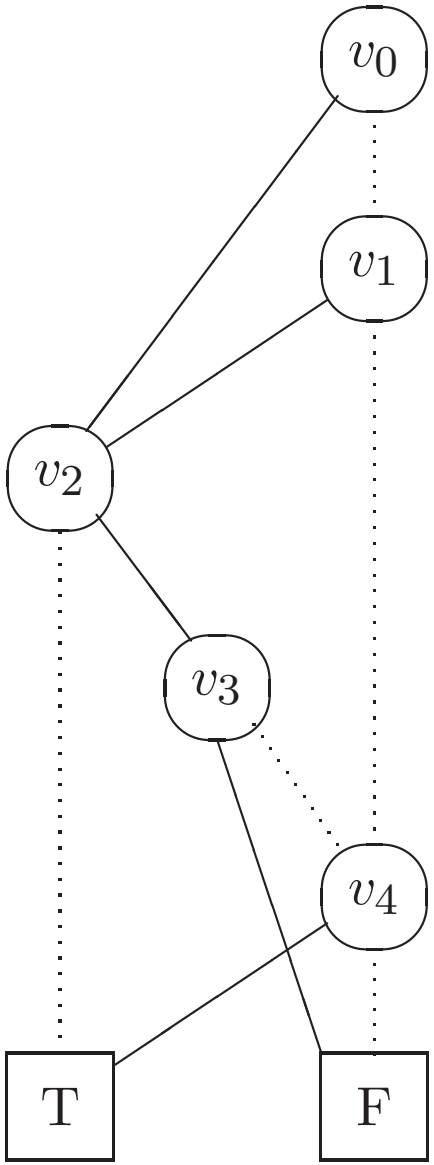
Example, Pigeon Hole Formulas

$$w_{i,j}^{n-1} \Leftrightarrow v_{i,j} \vee (v_{n,j} \wedge v_{i,n+1}), \qquad 1 \leq i \leq n-1,\ 1 \leq j \leq n$$

All the $w_{i,j}^{n-1}$ act like the $v_{i,j}$ except that the maximum of i and j are reduced by 1. That is, if a unique mapping is possible and the modified formula is satisfied, one of $w_{i,1}^{n-1}, w_{i,2}^{n-1}, \ldots, w_{i,n}^{n-1}$, $1 \leq i \leq n-1$, will have value 1 and all clauses $\overline{w_{i,j}^{n-1}} \vee \overline{w_{i,k}^{n-1}}$ will also have value 1.

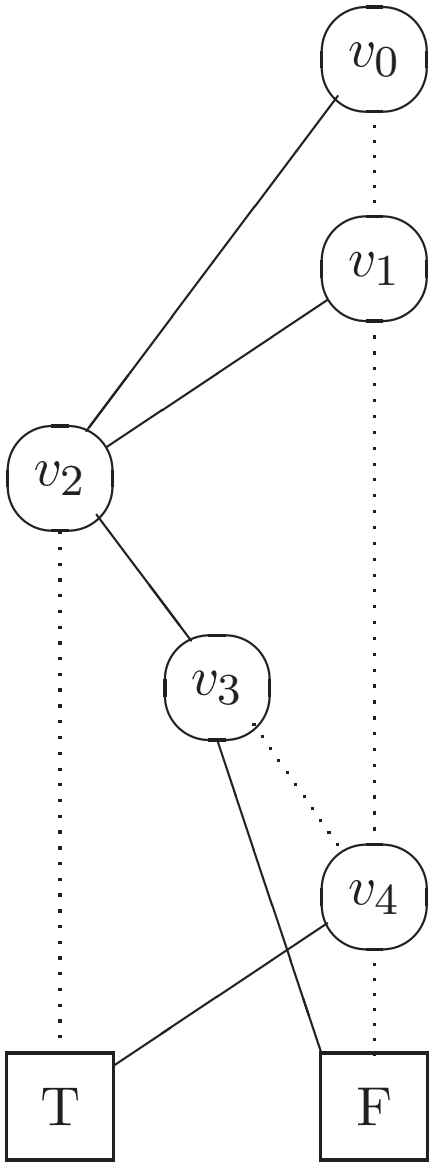


Binary Decision Diagrams



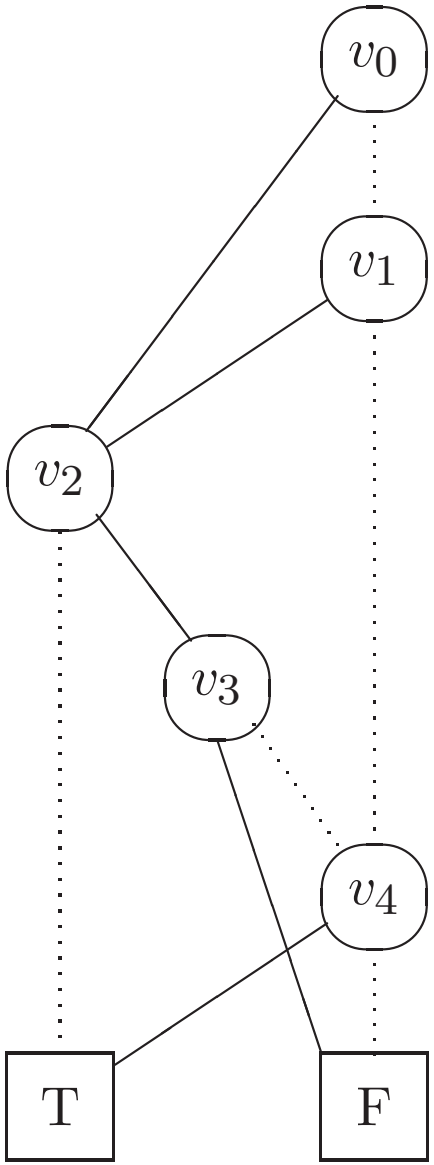
Rooted binary directed acyclic graph
Two leaves labeled T and F
Other nodes are labeled with variable names
Edges are dotted indicating a value of 0
or solid indicating a value of 1 for up var
Out of each non-leaf, there is one solid and one dotted edge
There is an order on the variables of the BDD and variables on any path obey it
A single BDD compactly represents a Boolean function

Binary Decision Diagrams



v_0	v_1	v_2	v_3	v_4	f
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	0	1	1
1	0	0	1	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	1
1	0	1	1	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	0	1	1
1	1	0	1	0	1
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	1
1	1	1	1	0	0
1	1	1	1	1	0

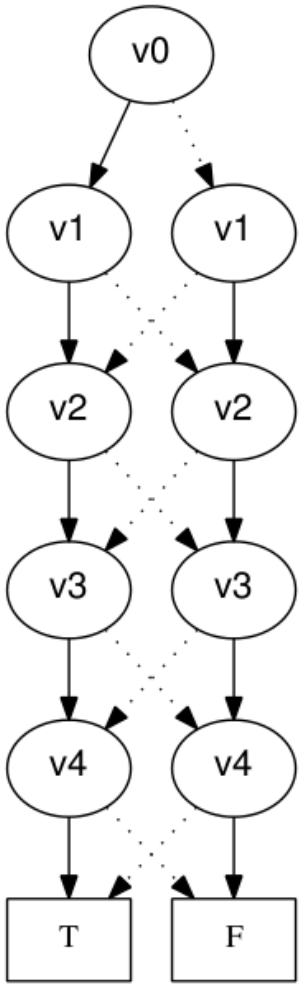
Binary Decision Diagrams



Rooted binary directed acyclic graph
Two leaves labeled T and F
Other nodes are labeled with variable names
Edges are dotted indicating a value of 0
or solid indicating a value of 1 for up var
Out of each non-leaf, there is one solid and one dotted edge
There is an order on the variables of the BDD and variables on any path obey it
A single BDD compactly represents a Boolean function
Every path to F represents a clause, anding them gives the function

$$(v_0 \vee v_1 \vee v_4) \wedge (v_0 \vee \overline{v_1} \vee \overline{v_2} \vee v_3 \vee v_4) \wedge (v_0 \vee \overline{v_1} \vee \overline{v_2} \vee \overline{v_3}) \wedge$$
$$(\overline{v_0} \vee \overline{v_2} \vee v_3 \vee v_4) \wedge (\overline{v_0} \vee \overline{v_2} \vee \overline{v_3})$$

Binary Decision Diagrams

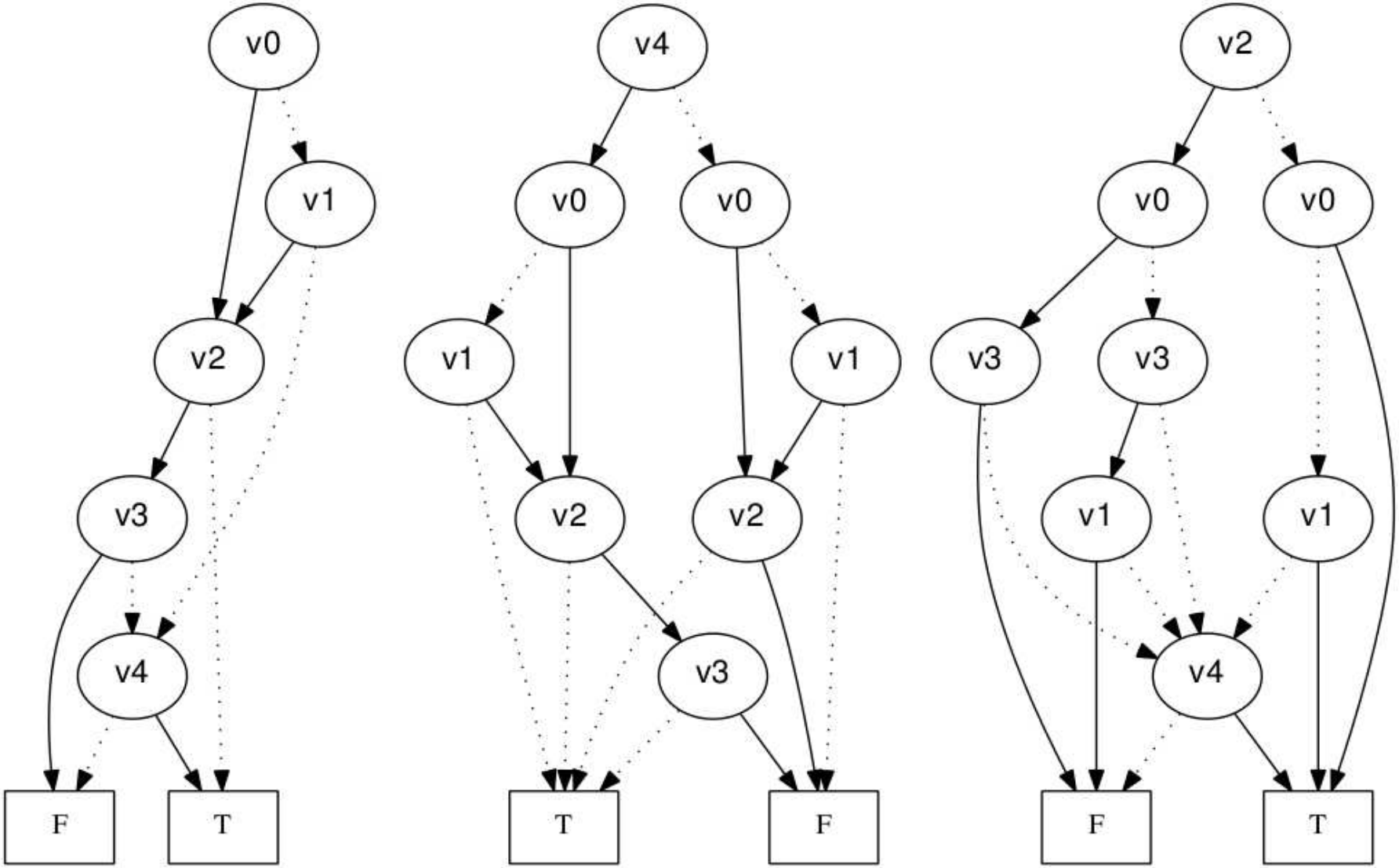


$$
\begin{aligned}
&(v_0 \vee v_1 \vee v_2 \vee v_3 \vee v_4) \wedge \\
&(\overline{v_0} \vee \overline{v_1} \vee v_2 \vee v_3 \vee v_4) \wedge \\
&(\overline{v_0} \vee v_1 \vee \overline{v_2} \vee v_3 \vee v_4) \wedge \\
&\quad \ldots \wedge \\
&(v_0 \vee v_1 \vee v_2 \vee \overline{v_3} \vee \overline{v_4}) \wedge \\
&\quad \ldots \wedge \\
&(\overline{v_0} \vee \overline{v_1} \vee \overline{v_2} \vee \overline{v_3} \vee v_4) \wedge \\
&(\overline{v_0} \vee \overline{v_1} \vee \overline{v_2} \vee v_3 \vee \overline{v_4}) \wedge \\
&\quad \ldots \wedge \\
&(\overline{v_0} \vee v_1 \vee \overline{v_2} \vee \overline{v_3} \vee \overline{v_4}) \wedge \\
&(v_0 \vee \overline{v_1} \vee \overline{v_2} \vee \overline{v_3} \vee \overline{v_4}) \wedge
\end{aligned}
$$

XOR representations: BDDs are exponentially smaller than CNFs unless you do something like this (for parity constraint):

$$(v_0 \oplus w_0) \Leftrightarrow p \wedge (v_1 \oplus w_1) \Leftrightarrow w_0 \wedge \ldots \wedge (v_{n-2} \oplus w_{n-1}) \Leftrightarrow w_{n-2} \wedge v_{n-1} \Leftrightarrow w_{n-1}$$

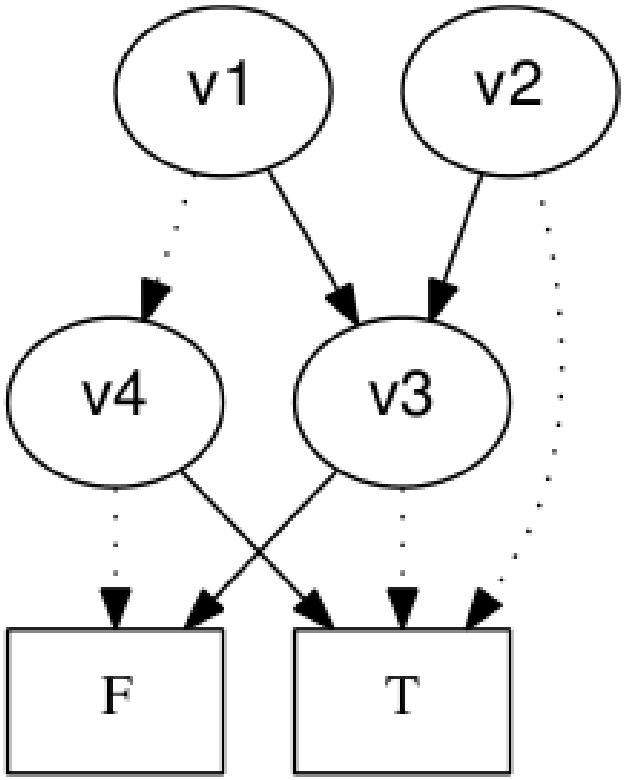
Binary Decision Diagrams



Via the BDD-visualizer - ordering affects size

available from: http://www.cs.uc.edu/~weaversa/BDD_Visualizer.html

Binary Decision Diagrams




```
order(v4, v3, v2, v1, v0)
ite(v4,T,F) ; BDD $1 Created
ite(v3,F,T) ; BDD $2 Created
ite(v2,$2,T) ; bdd $3 created
ite(v1,$2,$1) ; bdd $4 created
print($3,$4)
```

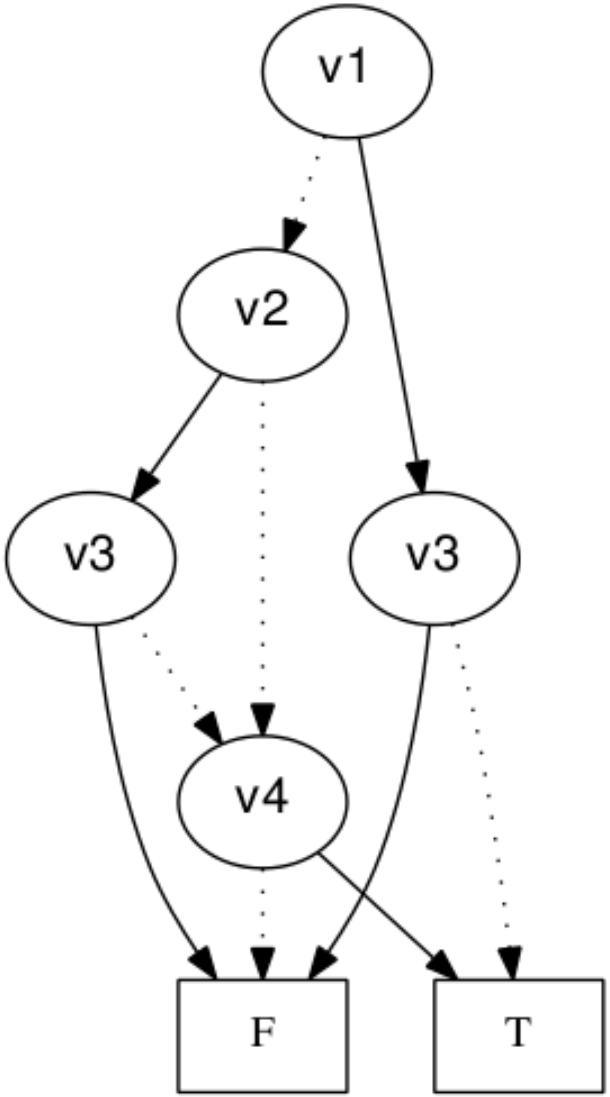
```
findOrCreateNode(v,t,e) {
    if (t == e) return t;
    if ((node = lookup(<v,t,e>)) != null)
        return node;
    node = createNode(<v,t,e>);
    insertNodeDataBase(<v,t,e>,node)
    return node;
}
```

Hashtable lookup on nodes - nodes shared across BDDs

(above is an example of the language of the BDD_Visualizer)

Binary Decision Diagrams

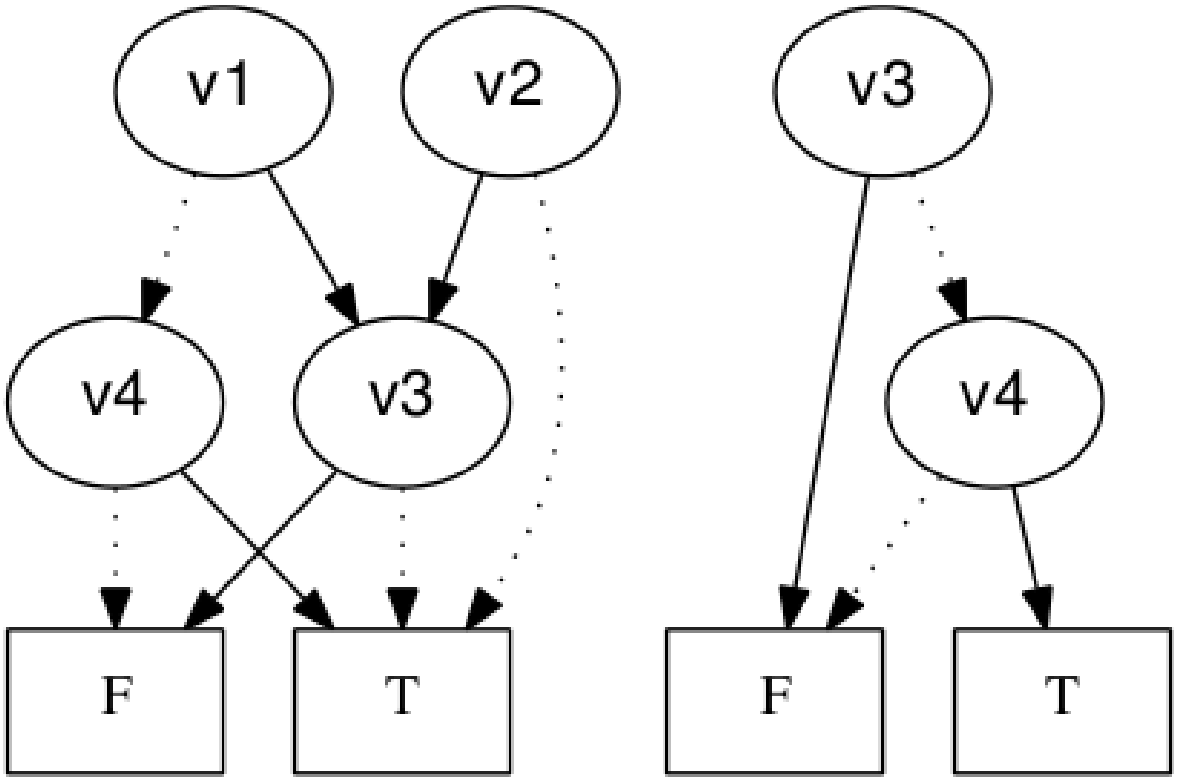
Op: Conjunction -




```
order(v4, v3, v2, v1, v0)
ite(v4,T,F) ; BDD $1 Created
ite(v3,F,T) ; BDD $2 Created
ite(v2,$2,T) ; bdd $3 created
ite(v1,$2,$1) ; bdd $4 created
and($3,$4) ; bdd $5 created
print($5)
```

All interesting applications conjoin BDDs

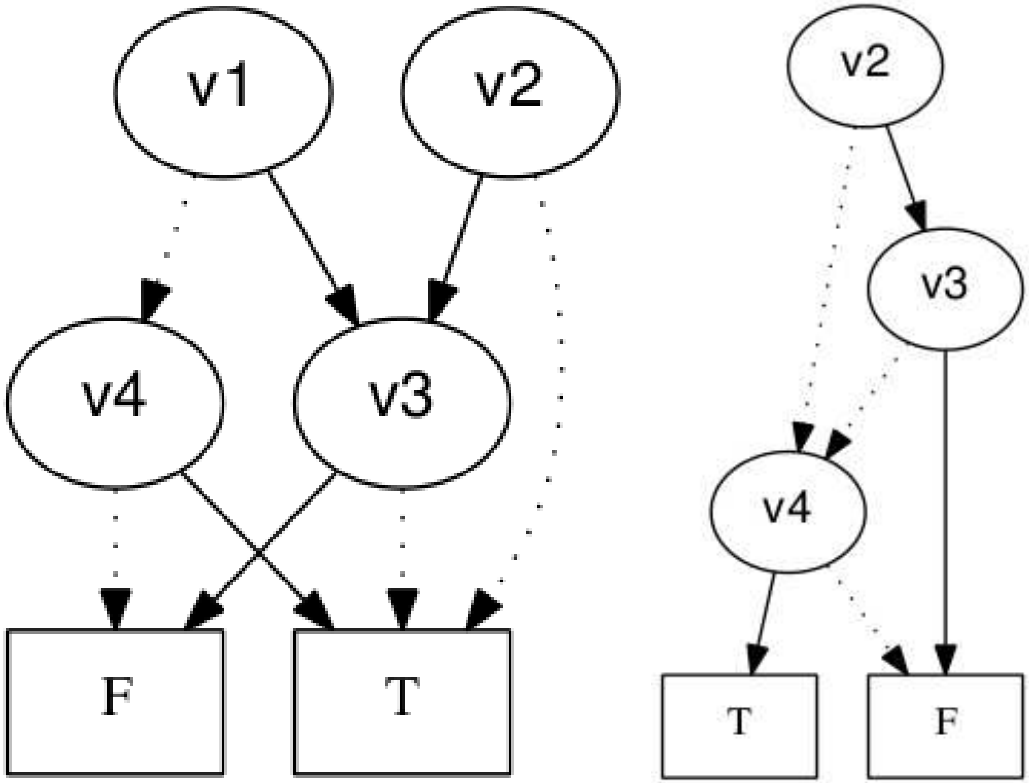
Binary Decision Diagrams



Conjoining is a bottom up process - complexity linear in # nodes

Both $v_3 = 0$ and $v_4 = 1$ to get to T

Binary Decision Diagrams

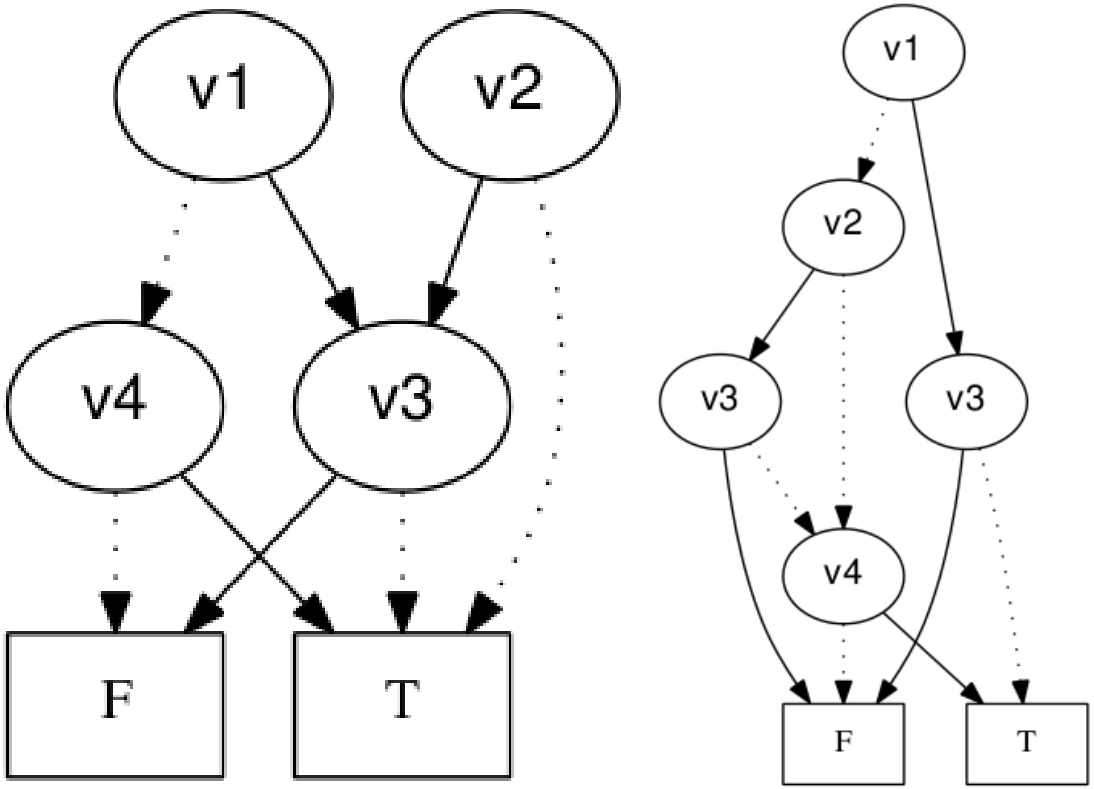


Conjoining is a bottom up process - complexity linear in # nodes

$v_2 = 1$ goes to v_3, no change

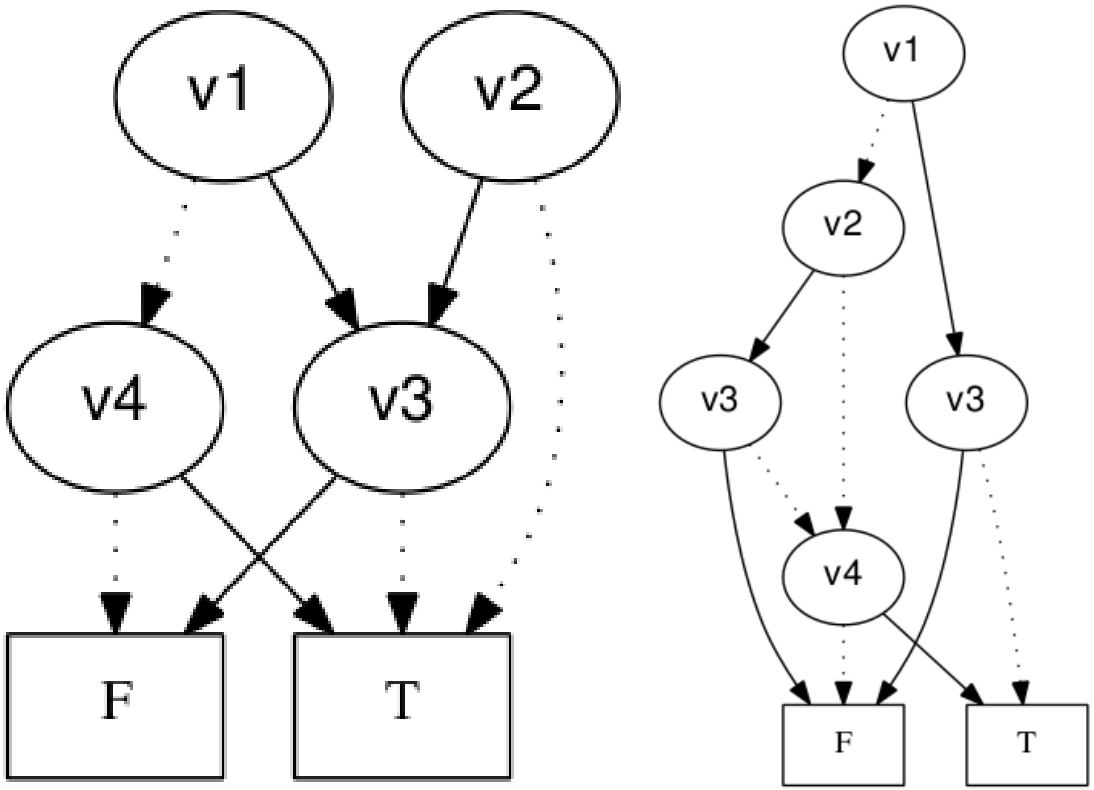
If $v_2 = 0$ then $v_4 = 1$ to get to T

Binary Decision Diagrams



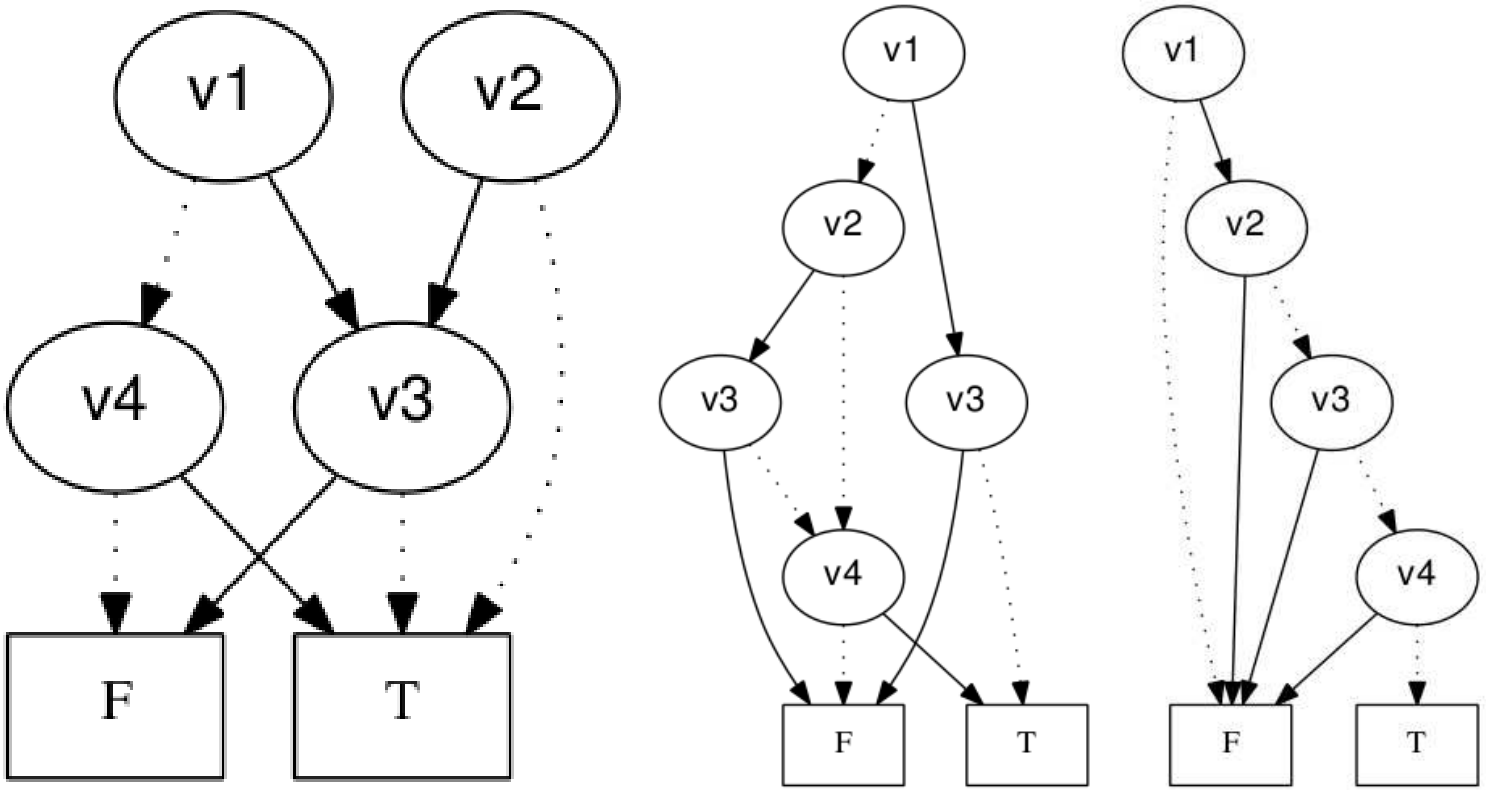
Conjoining is a bottom up process - complexity linear in # nodes

Binary Decision Diagrams



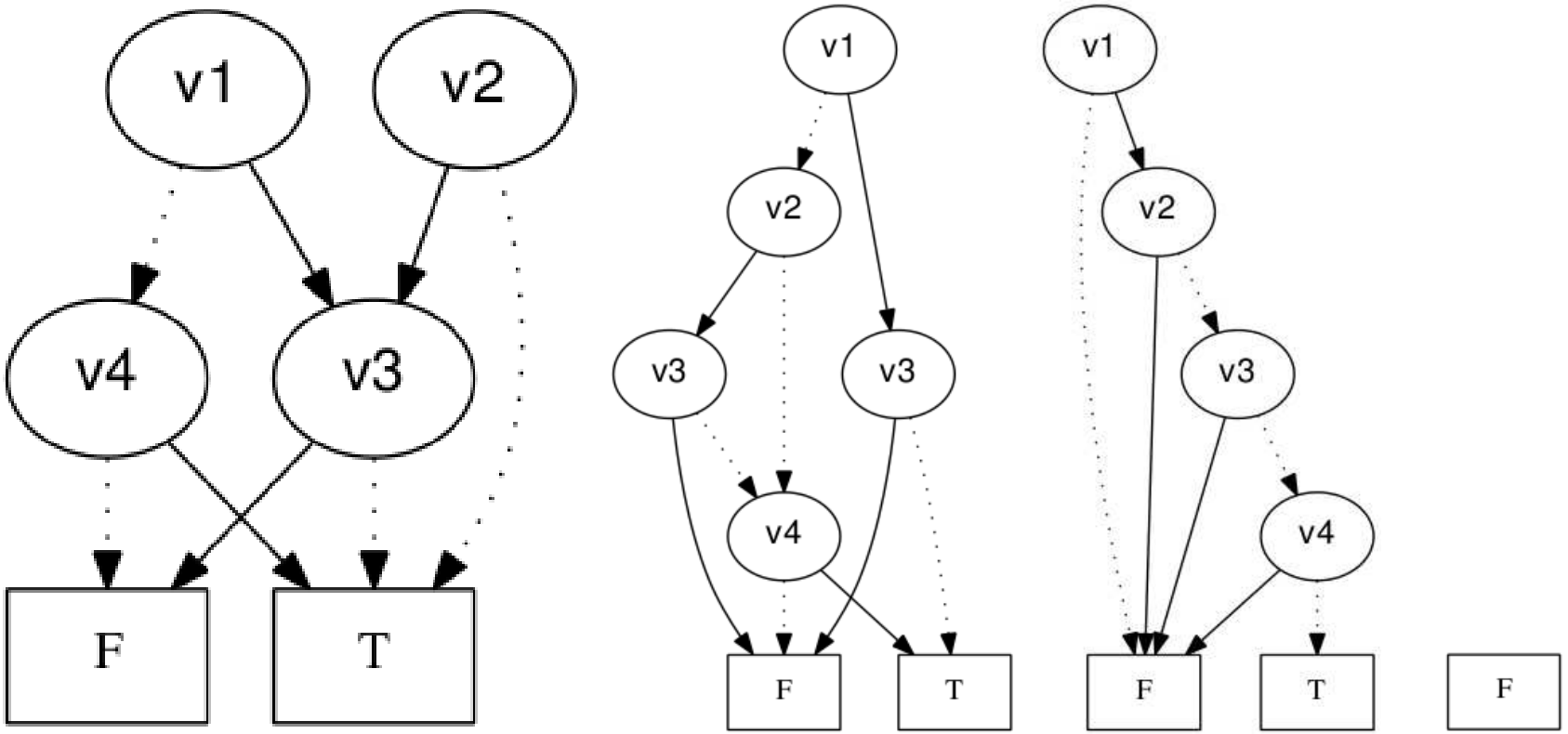
Conjoining is a bottom up process - complexity linear in # nodes but the number of nodes can double - hence 2^n nodes may result from n binary conjunctions.

Binary Decision Diagrams



Conjoining is a bottom up process - complexity linear in # nodes but the number of nodes can double - hence 2^n nodes may result from n binary conjunctions. But more conjunctions may reduce the size of the BDD - here `not(or(v2,v4))` is added.

Binary Decision Diagrams



Conjoining is a bottom up process - complexity linear in # nodes but the number of nodes can double - hence 2^n nodes may result from n binary conjunctions. But more conjunctions may reduce the size of the BDD - here `not(or(v2,v4))` is added. Then `or(v2,v3)`

Binary Decision Diagrams

Op: Existential quantification -

A Boolean function which can be written

$$f(v, \vec{x}) = (v \wedge h_1(\vec{x})) \vee (\overline{v} \wedge h_2(\vec{x}))$$

can be replaced by

$$f(\vec{x}) = h_1(\vec{x}) \vee h_2(\vec{x})$$

where $\vec{x}$ is a list of one or more variables.

There is a solution to $f(\vec{x})$ iff there is a solution to $f(v, \vec{x})$ so it is sufficient to solve $f(\vec{x})$ to get a solution to $f(v, \vec{x})$.

- a variable is eliminated!
- this is natural for BDDs

Binary Decision Diagrams

Op: Restrict -

Consider the truth tables for BDDs f and c.

Build a new BDD g over variables in f and c.

On any row of c's truth table that has value 1,
let the corresponding row in g map to
the same value as f.

On other rows g maps to any value.

Observe $f \wedge c$ and $g \wedge c$ are identical,
so g can replace f in a collection of BDDs.

BDD g is said to be a reduction -

inps	f
0000	0
0001	0
0010	1
0011	1
...	.

inps	c
0000	0
0001	1
0010	0
0011	1
...	.

inps	g
0000	1
0001	0
0010	0
0011	1
...	.

- BDD g can be made smaller than f.
- Inferences can be discovered.
- BDDs can be removed from the collection without loss.

Binary Decision Diagrams

Obvious restrict: g maps to 0 all rows that c maps to 0.

Called zero-restrict, has weaknesses - for example

$$c = (v_1 \vee \overline{v}_2) \wedge (\overline{v}_1 \vee \overline{v}_3)$$

$$f = (v_2 \vee \overline{v}_3)$$

will yield

$$g = \overline{v}_3 \wedge (v_1 \vee (\overline{v}_1 \wedge \overline{v}_2))$$

instead of the possible

$$g = (\overline{v}_3)$$

Obvious dual: g maps to 1 all rows that c maps to 0

No better.

Desired: some rows of g to map to 1, others to 0 so that

g's truth table reflects a pattern that generates inferences.

g maps $\langle 011 \rangle$ and $\langle 111 \rangle$ to 0 and $\langle 010 \rangle$ and $\langle 101 \rangle$ to 1 to get $\overline{v}_3$.

Binary Decision Diagrams

Op: Generalized Cofactor -

BDD g is a generalized co-factor of f and c if for any truth assignment t, $g(t)$ has the same value as $f(t')$ where t' is the nearest truth assignment to t that maps c to 1.

The notion of nearest truth assignment depends on a permutation π of the numbers 1,2,...,n that gives the variable ordering of the input BDDs.

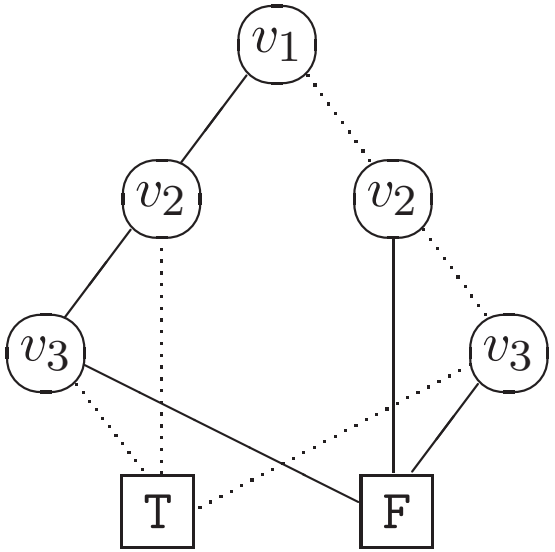
Represent a truth assignment to n variables as a vector in $\{0,1\}^n$ for assignment t, let t_i denote the i^{th} bit of the vector representing t. Then distance between two truth assignments t' and t'' is defined as

$$\sum_{i=1}^{n} 2^{n-i}(t'_{\pi_i} \oplus t''_{\pi_i}).$$

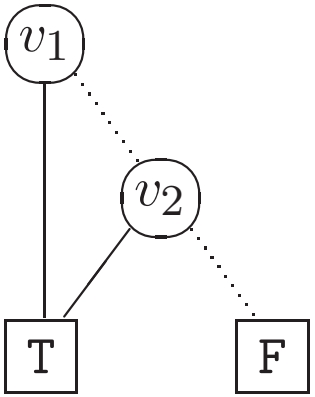
One pair of assignments is nearer to each other than another pair if the distance between that pair is less.

Binary Decision Diagrams

$f = (\overline{v}_1 \vee \overline{v}_3) \wedge (v_1 \vee \overline{v}_2) \wedge (v_1 \vee v_2 \vee \overline{v}_3)$



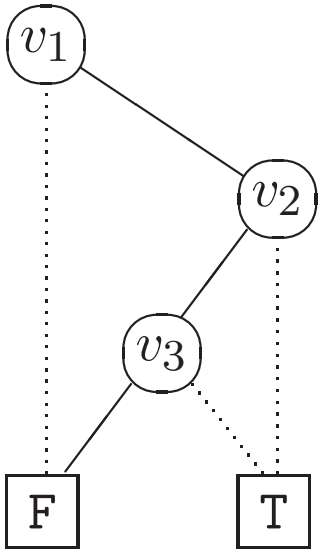
$c = (v_1 \vee v_2)$



inps	f
000	1
001	0
010	0
011	0
100	1
101	1
110	1
111	0

inps	c
000	0
001	0
010	1
011	1
100	1
101	1
110	1
111	1

$gcf(f, c) = (v_1 \wedge (v_2 \rightarrow \overline{v}_3))$



inps	g
000	0
001	0
010	0
011	0
100	1
101	1
110	1
111	0

Binary Decision Diagrams

Theorem:

Given BDDs $f_1, ..., f_k$, for any $1 \leq i \leq k$, $f_1 \wedge f_2 \wedge ... \wedge f_k$ is satisfiable if and only if $(f_1|f_i) \wedge ... \wedge (f_{i-1}|f_i) \wedge (f_{i+1}|f_i) \wedge ... \wedge (f_k|f_i)$ is satisfiable. Moreover, any assignment satisfying the latter can be mapped to an assignment that satisfies $f_1 \wedge ... \wedge f_k$.

So generalized co-factoring can be used to eliminate one of the BDDs among a given conjoined set of BDDs: the solver finds an assignment satisfying $\mathbf{gcf}(f_1, f_i) \wedge ... \wedge \mathbf{gcf}(f_k, f_i)$ and then extends the assignment to satisfy f_i, otherwise the solver reports that the instance has no solution.

Algebraic Methods

Example:

$$(v_1 \vee v_2 \vee v_3)$$

is represented by the equation

$$v_1(1+v_2)(1+v_3)+v_2(1+v_3)+v_3+1=0$$

which may be rewritten

$$v_1v_2v_3+v_1v_2+v_1v_3+v_2v_3+v_1+v_2+v_3+1=0$$

Example:

$$v_1 \oplus v_2 \oplus v_3 \oplus v_4$$

is represented by the equation

$$v_1+v_2+v_3+v_4+1=0.$$

Algebraic Methods

Arithmetic mod 2:

New facts are derived from old facts using the following rules:

1. Any even sum of like terms in an equation may be replaced by 0.

   e.g.: $v_1 v_2 + v_1 v_2 \Rightarrow 0$ and $1 + 1 \Rightarrow 0$.

   Needed to eliminate terms when adding equations.

2. A factor v^2 may be replaced by v
   Needed to ensure terms remain multi-linear after multiplication

3. An equation may be multiplied by a term,
   the resulting equation may be reduced by the rule above.

   e.g.: $v_3 v_4 (v_1 + v_3 = 0) \Rightarrow v_1 v_3 v_4 + v_3 v_4 = 0.$

   The new equation is said to be a new, derived fact.

4. Two equations may be added, using mod 2 arithmetic

   The new equation is said to be a new, derived fact.

Algebraic Methods

Example:

$$(v_1 \vee \overline{v}_2) \wedge (v_2 \vee \overline{v}_3) \wedge (v_3 \vee \overline{v}_1)$$

The equations corresponding to the above are shown below as equations (1), (2), and (3). All equations below the line are derived as stated on the right.

$$\begin{array}{llll}
v_1v_2 & +v_2 & = 0 & (1) \\
v_2v_3 & +v_3 & = 0 & (2) \\
v_1v_3 & +v_1 & = 0 & (3)
\end{array}$$

$$\begin{array}{llll}
v_1v_2v_3 & +v_2v_3 & = 0 & (4) \Leftarrow v_3 \cdot (1) \\
v_1v_2v_3 & +v_3 & = 0 & (5) \Leftarrow (4) + (2) \\
v_1v_2v_3 & +v_1v_3 & = 0 & (6) \Leftarrow v_1 \cdot (2) \\
v_1v_2v_3 & +v_1 & = 0 & (7) \Leftarrow (6) + (3) \\
v_1v_2v_3 + v_1v_2 & & = 0 & (8) \Leftarrow v_2 \cdot (3) \\
v_1v_2v_3 & +v_2 & = 0 & (9) \Leftarrow (8) + (1) \\
v_1 & +v_2 & = 0 & (10) \Leftarrow (9) + (7) \\
v_1 & +v_3 & = 0 & (11) \Leftarrow (5) + (7)
\end{array}$$

From the bottom two equations, $v_1 = v_2 = v_3$.

Algebraic Methods

An Algebraic Solver (ψ, d)

/* Input: List of equations $\psi = \langle e_1, ..., e_m \rangle$, integer d */
/* Output: "satisfiable" or "unsatisfiable" */
/* Locals: Set B of equations */

Set $B \leftarrow \emptyset$.
Repeat while $\psi \neq \emptyset$:
  Pop $e \leftarrow \psi$.
  Repeat while $\exists e' \in B : first\_non\text{-}zero(e) = first\_non\text{-}zero(e')$:
    Set $e \leftarrow reduce(e + e')$. /* Rule 4. */
    If e is $1 = 0$: Output "unsatisfiable"
    If e is not $0 = 0$:
      Set $B \leftarrow B \cup \{e\}$.
      If $degree(e) < d$:
        Repeat for all variables v:
          If $reduce(v \cdot e)$ has not been in ψ:
            Append $\psi \leftarrow reduce(v \cdot e)$. /* Rule 3. */
Output "satisfiable".

Algebraic Methods

Arithmetic mod 2:

Theorem: The number of derivations used by the algebraic solver is within a polynomial factor of the minimum number possible.

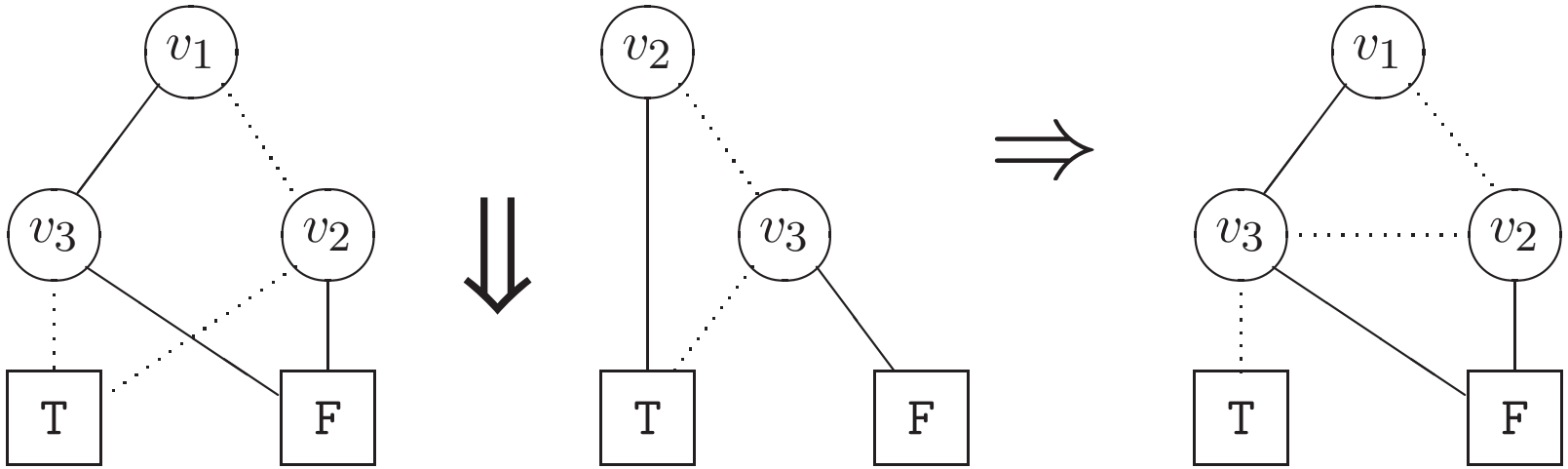
Theorem: The minimum number of derivations used by the algebraic solver cannot be much greater than, and may sometimes be far less than the minimum number needed by resolution.

Algebraic Methods

Arithmetic mod 2:

Comparison with BDD operations (`zero-restrict`(f,c)):

$f = (v_1 \vee \overline{v}_2) \wedge (\overline{v}_1 \vee \overline{v}_3)$ $\qquad$ $c = (v_2 \vee \overline{v}_3)$



Algebra: $f : v_1v_3 + v_2 + v_1v_2 = 0$ and $c : v_2v_3 + v_3 = 0$

$(v_2v_3) \cdot (v_1v_3 + v_2 + v_1v_2 = 0) \Rightarrow (v_2v_3 = 0) + (v_2v_3 + v_3 = 0) \Rightarrow v_3 = 0.$

As BDDs: multiply f by (v_2v_3) means conjoin f with $(\overline{v_2} \vee \overline{v_3})$.

Then $(\overline{v_2} \vee \overline{v_3})$ can be added to $f \wedge c$... since all 1 rows of f are 1 rows of $(\overline{v_2} \vee \overline{v_3})$.

But when $(\overline{v_2} \vee \overline{v_3})$ is conjoined with c, the inference $v_3 = 0$ is obtained.

Algebraic Methods

Arithmetic mod 2:

Comparison with BDD operations (gcf(f,c) and Ex. Quant.):

On BDDs, gcf(f,c) depends on the variable ordering

But gcf(f,c) may replace f, not so for algebra

In algebra, ex. quant. means multiply two equations

Example: consider $g = v_1v_2v_3 + v_1v_3 + v_1 + 1 = 0$.

To existentially quantify v_2 away from g: form equations:

$v_1 + 1 = 0 \quad (v_2 = 1)$

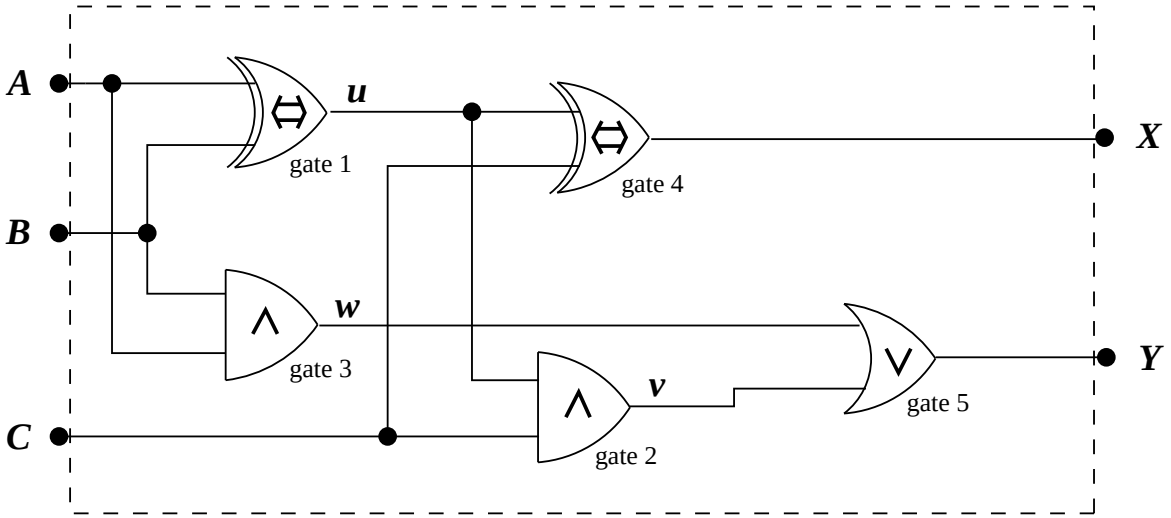
$v_1v_3 + v_1 + 1 = 0 \quad (v_2 = 0)$

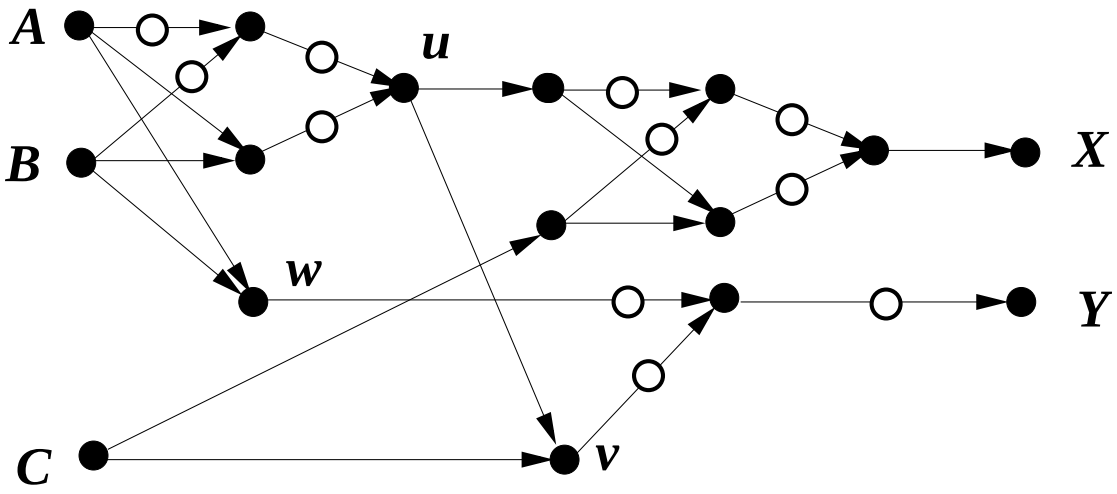
Then multiply

$$(v_1 + 1) \cdot (v_1v_3 + v_1 + 1) = (v_1v_3 + v_1v_3 + v_1 + 1) = (v_1 + 1).$$

But the variable that is quantified away can be in just one equation.

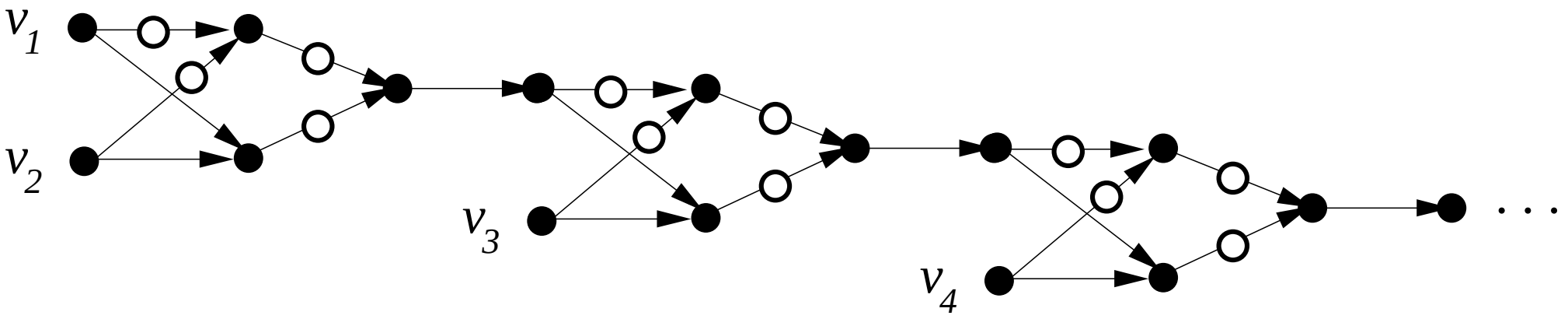
And Inverter Graphs





And Inverter Graphs

Can be small vs. CNF representation:



linear in AIG but exponential in CNF or in DNF

$$(v_1 \vee v_2 \vee v_3 ...) \wedge (\bar{v}_1 \vee \bar{v}_2 \vee v_3 ...) \wedge (\bar{v}_1 \vee v_2 \vee \bar{v}_3 ...) ... (\bar{v}_1 \vee \bar{v}_2 \vee \bar{v}_3 \vee \bar{v}_4 \vee v_5 ...) ...$$

And Inverter Graphs

Can be small vs. BDD representation:

Consider an integer multiplier for word size n

with outputs numbered 0 to $2n - 1$.

For the Boolean function representing either the output

$i - 1$ or $2n - i - 1$:

1. there is a circuit of size linear in n that implements it;
2. every BDD representing one of them has exponential size.

Satisfiability Modulo Theories

DPLL solves Satisfiability fine on some instances but not others

- Does not do well on proving multipliers correct
  - pigeon hole formulas
  - cardinality constraints
- Can do well on bounded model checking
  - but often it does not
- Is intended for propositional formulas

SMT combines subsolvers for certain classes of first order formulas with a DPLL SAT solver

Satisfiability Modulo Theories

What is a first order theory?

A set of first order formulas with no free variables

expressed in terms of specific function, predicate, constant, and variable symbols with some semantics arising from the functions, predicates, and values that the variables take.

A set of axioms is defined to derive expressions that are true in the theory.

Satisfiability Modulo Theories

Example theory: Linear Arithmetic

Equations or inequalities with addition/subtraction over rational or integer variables with constant multipliers

Syntax:

$$
\begin{aligned}
formula: &\quad formula \wedge formula \mid (formula) \mid atom \\
atom: &\quad sum\ op\ sum \\
op: &\quad \leq \mid = \mid < \\
sum: &\quad term \mid sum\ +\ term \mid sum\ -\ term \\
term: &\quad identifier \mid constant \mid constant\ identifier
\end{aligned}
$$

Examples:

$$x \leq y$$

$$x = 3y + z \wedge 2x - y < 0$$

Decision procedures:

CPLEX, gaussian elimination, branch-and-bound etc.

Satisfiability Modulo Theories

Example theory: Equalities & Uninterpreted Functions

Uninterpreted function and predicate symbols plus equals

Requires:

$x = y$ implies $f(x) = f(y)$

Examples:

$g(f(g(x))) \neq x$

$f(x) = x$

$p(x, y)$

$(x = y) \wedge (y = z) \wedge (f(x) \neq f(z))$

Satisfiability Modulo Theories

Example theory: Lists

Constraints support usual query, construction, and destruction.

Let x be a lisp-pair. Define

$\texttt{car}(x)$ to be the left element of x,

$\texttt{cdr}(x)$ to be the right element of x,

$\texttt{cons}(a, b)$ to be a lisp-pair with a on the left and b on the right,

$\texttt{atom}(a)$ to be true if a is not a lisp-pair.

Semantics:

$$\texttt{car}(\texttt{cons}(x, y)) = x$$

$$\texttt{cdr}(\texttt{cons}(x, y)) = y$$

$$\frac{}{\texttt{atom}(x) \rightarrow \texttt{cons}(\texttt{car}(x), \texttt{cdr}(x)) = x}$$

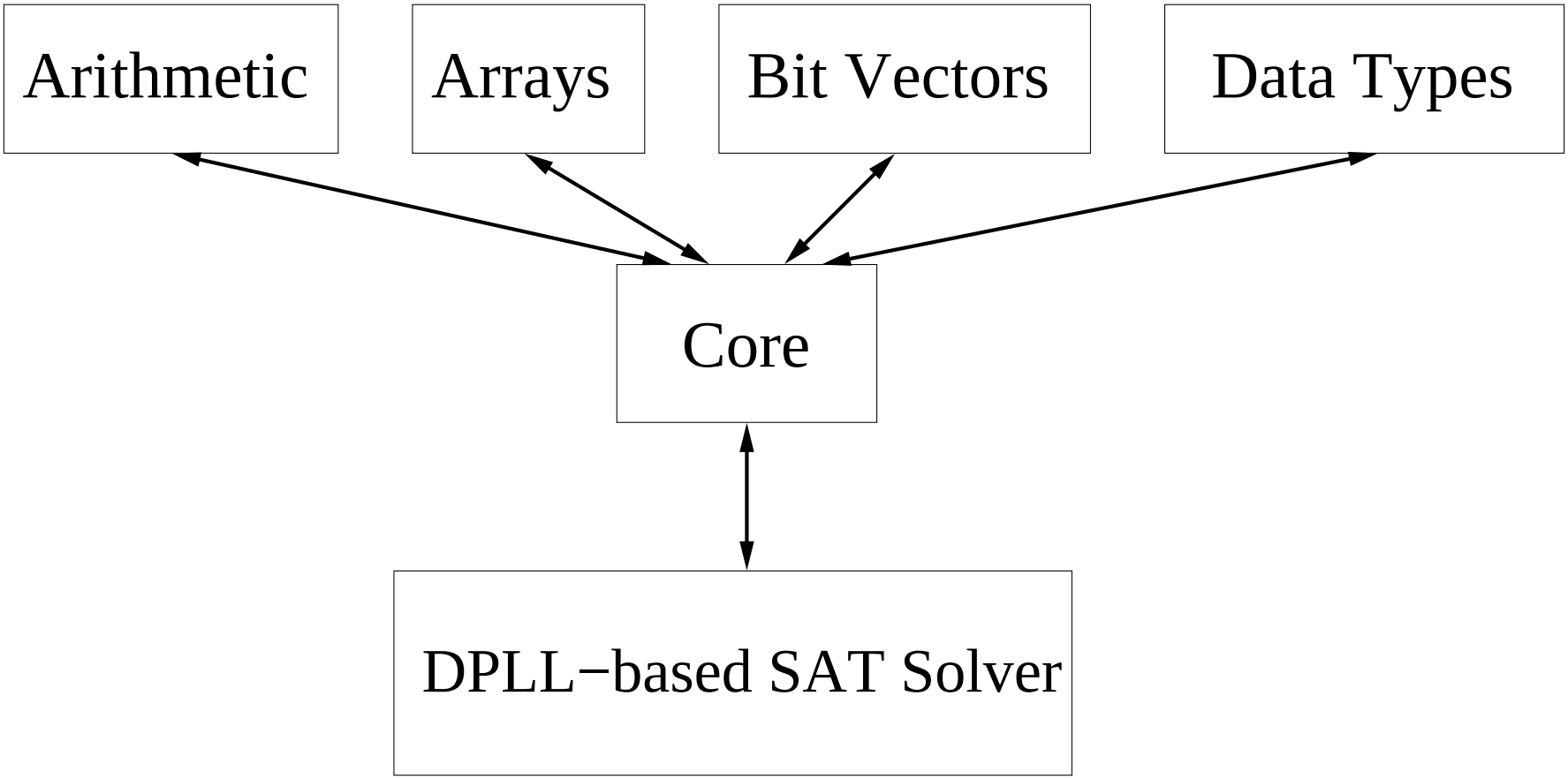
$$\frac{}{\texttt{atom}(\texttt{cons}(x, y))}$$

Satisfiability Modulo Theories

How does it work?

The SAT solver takes care of reasoning

When needed, it consults a theory solver which decides the validity of predicates.



Satisfiability Modulo Theories

Example:

A query - prove that

$$(x \leq y \wedge$$
$$y \leq x + \mathtt{car}(\mathtt{cons}(0, x)) \wedge$$
$$p(h(x) - h(y)) \rightarrow p(0)$$

where p is a function returning true or false

x and y are numbers, h is any function

Satisfiability Modulo Theories

Example:

A query - prove that

$$(x \leq y \wedge$$
$$y \leq x + \mathtt{car}(\mathtt{cons}(0, x)) \wedge$$
$$p(h(x) - h(y)) \rightarrow p(0)$$

where p is a function returning true or false

x and y are numbers, h is any function

The corresponding SMT sentence - show it's false

$$\phi = x \leq y;$$
$$y \leq x + \mathtt{car}(\mathtt{cons}(0, x));$$
$$p(h(x) - h(y));$$
$$\overline{p(0)};$$

Satisfiability Modulo Theories

$$\phi = x \leq y; \quad (1)$$
$$y \leq x + \mathtt{car}(\mathtt{cons}(0, x)); \quad (2)$$
$$p(h(x) - h(y)); \quad (3)$$
$$\overline{p(0)}; \quad (4)$$

Purify: create constants, add expressions

Satisfiability Modulo Theories

$$\phi = x \leq y; \quad (1)$$
$$y \leq x + \mathtt{car}(\mathtt{cons}(0, x)); \quad (2)$$
$$p(h(x) - h(y)); \quad (3)$$
$$\overline{p(0)}; \quad (4)$$

Purify: create constants, add expressions

(1): not mixed, leave alone

Satisfiability Modulo Theories

$$\phi = x \leq y; \quad (1)$$
$$y \leq x + \mathtt{car}(\mathtt{cons}(0, x)); \quad (2)$$
$$p(h(x) - h(y)); \quad (3)$$
$$\overline{p(0)}; \quad (4)$$

Purify: create constants, add expressions

$h(x) - h(y)$: mixed, create $g_1 = h(x)$, $g_2 = h(y)$

Satisfiability Modulo Theories

$$\phi = x \leq y; \quad (1)$$
$$y \leq x + \mathtt{car}(\mathtt{cons}(0, x)); \quad (2)$$
$$p(g_1 - g_2); \quad (3)$$
$$\overline{p(0)}; \quad (4)$$
$$g_1 = h(x);$$
$$g_2 = h(y);$$

Purify: create constants, add expressions

(3): is now $p(g_1 - g_2)$, mixed, create $g_3 = g_1 - g_2$

Satisfiability Modulo Theories

$$\phi = x \leq y; \quad (1)$$
$$y \leq x + \mathtt{car}(\mathtt{cons}(0, x)); \quad (2)$$
$$p(g_3); \quad (3)$$
$$\overline{p(0)}; \quad (4)$$
$$g_1 = h(x);$$
$$g_2 = h(y);$$
$$g_3 = g_1 - g_2;$$

Purify: create constants, add expressions

(3): becomes $p(g_3)$, not mixed, leave alone

Satisfiability Modulo Theories

$$\phi = x \leq y; \quad (1)$$
$$y \leq x + \mathtt{car}(\mathtt{cons}(0, x)); \quad (2)$$
$$p(g_3); \quad (3)$$
$$\overline{p(0)}; \quad (4)$$
$$g_1 = h(x);$$
$$g_2 = h(y);$$
$$g_3 = g_1 - g_2;$$

Purify: create constants, add expressions

0 alien in $p(0)$: create $g_4 = 0$

Satisfiability Modulo Theories

$$\phi = x \leq y; \quad (1)$$
$$y \leq x + \mathtt{car}(\mathtt{cons}(g_4, x)); \quad (2)$$
$$p(g_3); \quad (3)$$
$$\overline{p(g_4)}; \quad (4)$$
$$g_1 = h(x);$$
$$g_2 = h(y);$$
$$g_3 = g_1 - g_2;$$
$$g_4 = 0;$$

Purify: create constants, add expressions

(4): becomes $\overline{p(g_4)}$, not mixed, leave alone

Satisfiability Modulo Theories

$$\phi = x \leq y; \quad (1)$$
$$y \leq x + \mathtt{car}(\mathtt{cons}(g_4, x)); \quad (2)$$
$$p(g_3); \quad (3)$$
$$\overline{p(g_4)}; \quad (4)$$
$$g_1 = h(x);$$
$$g_2 = h(y);$$
$$g_3 = g_1 - g_2;$$
$$g_4 = 0;$$

Purify: create constants, add expressions

(2): mixed, create $g_5 = \mathtt{car}(\mathtt{cons}(g_4, x))$

Satisfiability Modulo Theories

$$\phi = x \leq y; \quad (1)$$
$$y \leq x + g_5; \quad (2)$$
$$p(g_3); \quad (3)$$
$$\overline{p(g_4)}; \quad (4)$$
$$g_1 = h(x);$$
$$g_2 = h(y);$$
$$g_3 = g_1 - g_2;$$
$$g_4 = 0;$$
$$g_5 = \texttt{car}(\texttt{cons}(g_4, x));$$

Purify: create constants, add expressions

(2): becomes $y \leq x + g_5$, leave alone

Satisfiability Modulo Theories

$\phi_1 = x \leq y;\ y \leq x + g_5;\ g_3 = g_1 - g_2;\ g_4 = 0;$ (LA)

$\phi_2 = g_1 = h(x);\ g_2 = h(y);\ \overline{p(g_4)};\ p(g_3);$ (EUF)

$\phi_3 = g_5 = \texttt{car}(\texttt{cons}(g_4, x));$ (Lists)

Propagate inferred equalities between ϕ_1, ϕ_2, ϕ_3

Satisfiability Modulo Theories

$\phi_1 = x \leq y;\ y \leq x + g_5;\ g_3 = g_1 - g_2;\ g_4 = 0;$ (LA)

$\phi_2 = g_1 = h(x);\ g_2 = h(y);\ \overline{p(g_4)};\ p(g_3);$ (EUF)

$\phi_3 = g_5 = \texttt{car}(\texttt{cons}(g_4, x));$ (Lists)

Propagate inferred equalities between ϕ_1, ϕ_2, ϕ_3

$g_5 = \texttt{car}(\texttt{cons}(g_4, x))$ infers $g_4 = g_5$ (List axiom)

Satisfiability Modulo Theories

$\phi_1 = x \leq y;\ y \leq x + g_5;\ g_3 = g_1 - g_2;\ g_5 = 0;$ (LA)

$\phi_2 = g_1 = h(x);\ g_2 = h(y);\ \overline{p(g_5)};\ p(g_3);$ (EUF)

$\phi_3 = g_5 = \mathtt{car}(\mathtt{cons}(g_5, x));$ (Lists)

Propagate inferred equalities between ϕ_1, ϕ_2, ϕ_3

$g_5 = \mathtt{car}(\mathtt{cons}(g_4, x))$ infers $g_4 = g_5$ (List axiom)

Satisfiability Modulo Theories

$\phi_1 = x \leq y;\ y \leq x + g_5;\ g_3 = g_1 - g_2;\ g_5 = 0;$ (LA)

$\phi_2 = g_1 = h(x);\ g_2 = h(y);\ \overline{p(g_5)};\ p(g_3);$ (EUF)

$\phi_3 = g_5 = \texttt{car}(\texttt{cons}(g_5, x));$ (Lists)

Propagate inferred equalities between ϕ_1, ϕ_2, ϕ_3

$g_5 = \texttt{car}(\texttt{cons}(g_4, x))$ infers $g_4 = g_5$ (List axiom)

$g_5 = 0$ is inferred in ϕ_1 (Equals for equals)

Satisfiability Modulo Theories

$\phi_1 = x \leq y;\ y \leq x + g_5;\ g_3 = g_1 - g_2;\ g_5 = 0;$ (LA)

$\phi_2 = g_1 = h(x);\ g_2 = h(y);\ \overline{p(g_5)};\ p(g_3);$ (EUF)

$\phi_3 = g_5 = \mathtt{car}(\mathtt{cons}(g_5, x));$ (Lists)

Propagate inferred equalities between ϕ_1, ϕ_2, ϕ_3

$g_5 = \mathtt{car}(\mathtt{cons}(g_4, x))$ infers $g_4 = g_5$ (List axiom)

$g_5 = 0$ is inferred in ϕ_1 (Equals for equals)

$x \leq y$ and $y \leq x + g_5$ infer $x = y$ (Linear Arithmetic axiom)

Satisfiability Modulo Theories

$\phi_1 = x = y;\ g_3 = g_1 - g_2;\ g_5 = 0;$ (LA)

$\phi_2 = g_1 = h(x);\ g_2 = h(x);\ \overline{p(g_5)};\ p(g_3);$ (EUF)

$\phi_3 = g_5 = \mathtt{car}(\mathtt{cons}(g_5, x));$ (Lists)

Propagate inferred equalities between ϕ_1, ϕ_2, ϕ_3

$g_5 = \mathtt{car}(\mathtt{cons}(g_4, x))$ infers $g_4 = g_5$ (List axiom)

$g_5 = 0$ is inferred in ϕ_1 (Equals for equals)

$x \leq y$ and $y \leq x + g_5$ infer $x = y$ (Linear Arithmetic axiom)

Satisfiability Modulo Theories

$\phi_1 = x = y;\ g_3 = g_1 - g_2;\ g_5 = 0;$ (LA)

$\phi_2 = g_1 = h(x);\ g_2 = h(x);\ \overline{p(g_5)};\ p(g_3);$ (EUF)

$\phi_3 = g_5 = \mathtt{car}(\mathtt{cons}(g_5, x));$ (Lists)

Propagate inferred equalities between ϕ_1, ϕ_2, ϕ_3

$g_5 = \mathtt{car}(\mathtt{cons}(g_4, x))$ infers $g_4 = g_5$ (List axiom)

$g_5 = 0$ is inferred in ϕ_1 (Equals for equals)

$x \leq y$ and $y \leq x + g_5$ infer $x = y$ (Linear Arithmetic axiom)

$g_1 = h(x)$ and $g_2 = h(x)$ infer $g_1 = g_2$ (EUF axiom)

Satisfiability Modulo Theories

$\phi_1 = x = y;\ g_3 = g_1 - g_1;\ g_5 = 0;$ (LA)

$\phi_2 = g_1 = h(x);\ g_1 = h(x);\ \overline{p(g_5)};\ p(g_3);$ (EUF)

$\phi_3 = g_5 = \texttt{car}(\texttt{cons}(g_5, x));$ (Lists)

Propagate inferred equalities between ϕ_1, ϕ_2, ϕ_3

$g_5 = \texttt{car}(\texttt{cons}(g_4, x))$ infers $g_4 = g_5$ (List axiom)

$g_5 = 0$ is inferred in ϕ_1 (Equals for equals)

$x \leq y$ and $y \leq x + g_5$ infer $x = y$ (Linear Arithmetic axiom)

$g_1 = h(x)$ and $g_2 = h(x)$ infer $g_1 = g_2$ (EUF axiom)

Satisfiability Modulo Theories

$\phi_1 = x = y;\ g_3 = g_1 - g_1;\ g_5 = 0;$ (LA)

$\phi_2 = g_1 = h(x);\ g_1 = h(x);\ \overline{p(g_5)};\ p(g_3);$ (EUF)

$\phi_3 = g_5 = \texttt{car}(\texttt{cons}(g_5, x));$ (Lists)

Propagate inferred equalities between ϕ_1, ϕ_2, ϕ_3

$g_5 = \texttt{car}(\texttt{cons}(g_4, x))$ infers $g_4 = g_5$ (List axiom)

$g_5 = 0$ is inferred in ϕ_1 (Equals for equals)

$x \leq y$ and $y \leq x + g_5$ infer $x = y$ (Linear Arithmetic axiom)

$g_1 = h(x)$ and $g_2 = h(x)$ infer $g_1 = g_2$ (EUF axiom)

$g_3 = g_1 - g_1$ infers $g_3 = 0 = g_5$ (LA axiom)

Satisfiability Modulo Theories

$\phi_1 = x = y;\ g_5 = g_1 - g_1;\ g_5 = 0;$ (LA)

$\phi_2 = g_1 = h(x);\ g_1 = h(x);\ \overline{p(g_5)};\ p(g_5);$ (EUF)

$\phi_3 = g_5 = \texttt{car}(\texttt{cons}(g_5, x));$ (Lists)

Propagate inferred equalities between ϕ_1, ϕ_2, ϕ_3

$g_5 = \texttt{car}(\texttt{cons}(g_4, x))$ infers $g_4 = g_5$ (List axiom)

$g_5 = 0$ is inferred in ϕ_1 (Equals for equals)

$x \leq y$ and $y \leq x + g_5$ infer $x = y$ (Linear Arithmetic axiom)

$g_1 = h(x)$ and $g_2 = h(x)$ infer $g_1 = g_2$ (EUF axiom)

$g_3 = g_1 - g_1$ infers $g_3 = 0 = g_5$ (LA axiom)

Satisfiability Modulo Theories

$\phi_1 = x = y;\ g_5 = g_1 - g_1;\ g_5 = 0;$ (LA)

$\phi_2 = g_1 = h(x);\ g_1 = h(x);\ \overline{p(g_5)};\ p(g_5);$ (EUF)

$\phi_3 = g_5 = \mathtt{car}(\mathtt{cons}(g_5, x));$ (Lists)

Propagate inferred equalities between ϕ_1, ϕ_2, ϕ_3

$g_5 = \mathtt{car}(\mathtt{cons}(g_4, x))$ infers $g_4 = g_5$ (List axiom)

$g_5 = 0$ is inferred in ϕ_1 (Equals for equals)

$x \leq y$ and $y \leq x + g_5$ infer $x = y$ (Linear Arithmetic axiom)

$g_1 = h(x)$ and $g_2 = h(x)$ infer $g_1 = g_2$ (EUF axiom)

$g_3 = g_1 - g_1$ infers $g_3 = 0 = g_5$ (LA axiom)

$\overline{p(g_5)} \wedge p(g_5)$ is a contradiction

Is Probabilistic Analysis Worthwhile?

The Questions:

- Why are some instances so difficult?
- Are there algorithms that will make them easier?

Is Probabilistic Analysis Worthwhile?

The Questions:

- Why are some instances so difficult?
- Are there algorithms that will make them easier?

Why Probability?

- Results and process tend to draw out intuition
  - Identify properties that may be exploited by a fast algorithm and properties that may prevent exploitation.
  - Identify reasons for the hardness of various instances - why **lots** of instances are hard.
- Can explain the good or bad behavior of an algorithm
- Afford comparison of incomparable classes of formulas

Is Probabilistic Analysis Worthwhile?

Some problems:

- Must assume an input distribution,
  often not reflecting reality
  (but sometimes we do not need to)
- Analysis can be difficult or impossible -
  algorithmic steps may significantly change distribution
  (known tools are limited)
- Can yield misleading results

A Misleading Result

Example: A probabilistic model for random formulas

Given: set $L = \{v_1, \overline{v_1}, ..., v_n, \overline{v_n}\}$ of literals and $0 < p < 1$

Construct clause c: $l \in L$ independently in c with probability p

Construct formula ψ: m independently constructed clauses

Justification: All formulas are equally likely if $p = 1/3$.

A Misleading Result

Example: A probabilistic model for random formulas

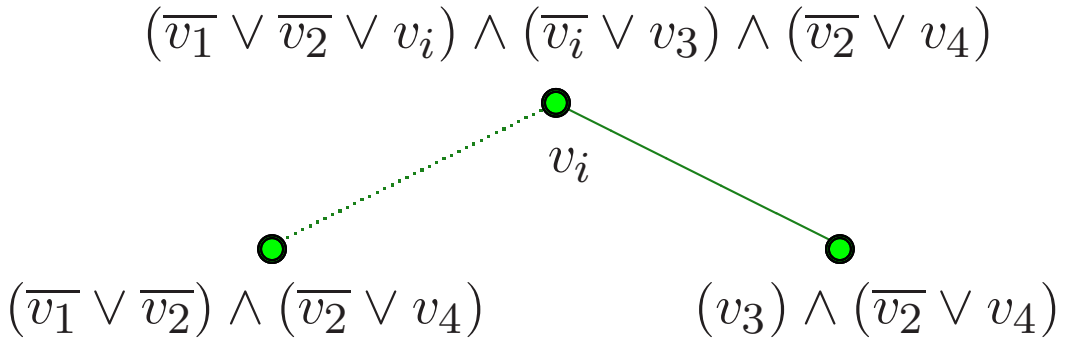
Given: set $L = \{v_1, \overline{v_1}, ..., v_n, \overline{v_n}\}$ of literals and $0 < p < 1$

Construct clause c: $l \in L$ independently in c with probability p

Construct formula ψ: m independently constructed clauses

Justification: All formulas are equally likely if $p = 1/3$.

Applied to splitting (DPLL):



A Misleading Result

Analysis sketch:

Given m clauses, the average number of clauses removed when a value is assigned is pm

A Misleading Result

Analysis sketch:

Given m clauses, the average number of clauses removed when a value is assigned is pm

Let T_i be the average number of clauses remaining on the i^{th} iteration

Then $T_0 = m$ and $T_i = (1 - p)T_{i-1}$

For what i does $T_i = 1$?

A Misleading Result

Analysis sketch:

Given m clauses, the average number of clauses removed when a value is assigned is pm

Let T_i be the average number of clauses remaining on the i^{th} iteration

Then $T_0 = m$ and $T_i = (1-p)T_{i-1}$

For what i does $T_i = 1$? When $1 = m(1-p)^i$ or

$$\lg(m) = -i \lg(1-p)$$
$$i = \lg(m)/-\lg(1-p)$$

So, size of search space is $2^i \approx 2^{\lg(m)/p} = m^c$ if $p = 1/3$

A Misleading Result

Analysis sketch:

Given m clauses, the average number of clauses removed when a value is assigned is pm

Let T_i be the average number of clauses remaining on the i^{th} iteration

Then $T_0 = m$ and $T_i = (1-p)T_{i-1}$

For what i does $T_i = 1$? When $1 = m(1-p)^i$ or

$$\lg(m) = -i \lg(1-p)$$
$$i = \lg(m)/ - \lg(1-p)$$

So, size of search space is $2^i \approx 2^{\lg(m)/p} = m^c$ if $p = 1/3$

Conclusion: *DPLL is fantastic, on the average!*

A Misleading Result

Problems with the analysis:

- The input model is funky!
  The probability that a random assignment satisfies a random formula is

$$(1-(1-p)^{2n})^m \approx e^{-me^{-2pn}}$$

  which tends to 1 if $\ln(m)/pn = o(1)$ and means the average width of a clause can be at least $3\ln(m)$.

  If all clauses have width $3\ln(m)$, max # assignments that cannot be models is at most $m2^n/2^{3\ln(m)} < (m/m)2^n$

A Misleading Result

Problems with the analysis:

- The input model is funky!
  The probability that a random assignment satisfies a random formula is

$$(1-(1-p)^{2n})^m \approx e^{-me^{-2pn}}$$

  which tends to 1 if $\ln(m)/pn = o(1)$ and means the average width of a clause can be at least $3\ln(m)$.

  If all clauses have width $3\ln(m)$, max # assignments that cannot be models is at most $m2^n/2^{3\ln(m)} < (m/m)2^n$

- If average clause width is constant $(p = c/n)$ then the average search space size is

$$2^{-\lg(m)/\lg(1-p)} \approx 2^{\lg(m)/(c/n)} = 2^{n\lg(m)/c} = m^{n/c}$$

A Misleading Result

Problems with the analysis:

- The input model is funky!
  The probability that a random assignment satisfies a random formula is

$$(1-(1-p)^{2n})^m \approx e^{-me^{-2pn}}$$

  which tends to 1 if $\ln(m)/pn = o(1)$ and means the average width of a clause can be at least $3\ln(m)$.

  If all clauses have width $3\ln(m)$, max # assignments that cannot be models is at most $m2^n/2^{3\ln(m)} < (m/m)2^n$

- If average clause width is constant $(p = c/n)$ then the average search space size is

$$2^{-\lg(m)/\lg(1-p)} \approx 2^{\lg(m)/(c/n)} = 2^{n\lg(m)/c} = m^{n/c}$$

Exponential complexity!

Probabilistic Toolbox

Linearity of expectation:

$$E\{\sum_i X_i\} = \sum_i E\{X_i\}, \quad X_i \text{ real valued r.v.s}$$

Probabilistic Toolbox

Linearity of expectation:

$$E\{\sum_i X_i\} = \sum_i E\{X_i\}, \quad X_i \text{ real valued r.v.s}$$

First Moment Method: show $Pr(P) \to 0$ as $n \to \infty$.

Let X be a count of some entity

Suppose some property P holds if and only if $X > 1$.

Since

$$Pr(X > 1) \leq E\{X\}$$

if $E\{X\} \to 0$ then $Pr(P) \to 0$, as $n \to \infty$.

First Moment Method

An Example:

Assume ψ is a random k-SAT formula, m clauses, n variables

Let P be the property that ψ has a model

First Moment Method

An Example:

Assume ψ is a random k-SAT formula, m clauses, n variables

Let P be the property that ψ has a model

Let X be the number of models for ψ

Let $X_i = \begin{cases} 1 & \text{if the } i^{th} \text{ assignment is a model for } \psi \\ 0 & \text{otherwise} \end{cases}$

First Moment Method

An Example:

Assume ψ is a random k-SAT formula, m clauses, n variables

Let P be the property that ψ has a model

Let X be the number of models for ψ

Let $X_i = \begin{cases} 1 & \text{if the } i^{th} \text{ assignment is a model for } \psi \\ 0 & \text{otherwise} \end{cases}$

$$Pr(X_i = 1) = (1 - 2^{-k})^m$$

$$E\{X\} = \sum_{i=1}^{2^n} Pr(X_i = 1) = 2^n (1 - 2^{-k})^m$$

First Moment Method

An Example:

Assume ψ is a random k-SAT formula, m clauses, n variables

Let P be the property that ψ has a model

Let X be the number of models for ψ

Let $X_i = \begin{cases} 1 & \text{if the } i^{th} \text{ assignment is a model for } \psi \\ 0 & \text{otherwise} \end{cases}$

$$Pr(X_i = 1) = (1 - 2^{-k})^m$$

$$E\{X\} = \sum_{i=1}^{2^n} Pr(X_i = 1) = 2^n(1 - 2^{-k})^m$$

$Pr(\psi \text{ has a model}) \to 0$ if $\frac{m}{n} > \frac{1}{\lg(1 - 2^{-k})} \approx 2^k$

For $k = 3$, ψ has no model w.h.p. if $\frac{m}{n} > 5.19$

Probabilistic Toolbox

Flow Analysis:

Consider straight-line (non-backtracking) variants of DPLL

Let ψ be a CNF Boolean expression
Set $M = \emptyset$
Repeat the following:
    Choose an unassigned literal l in ψ
    If l is a positive literal, set $M = M \cup \{l\}$
    Remove all clauses containing l from ψ
    Remove all literals $\bar{l}$ from ψ
    If ψ is empty then return M
    If some clause in ψ is falsified return "give up"

Probabilistic Toolbox

Flow Analysis:

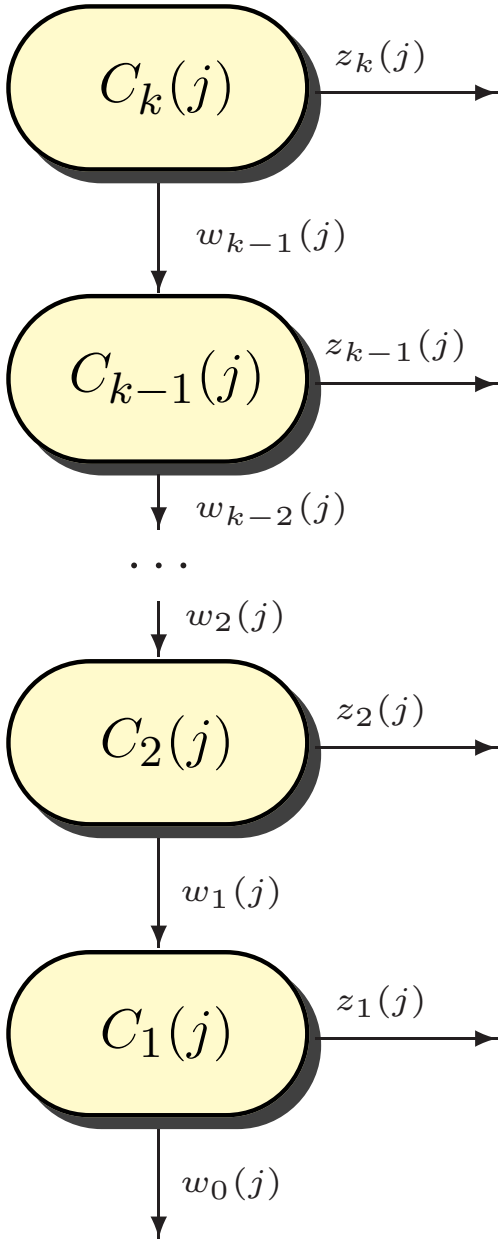
Consider straight-line (non-backtracking) variants of DPLL

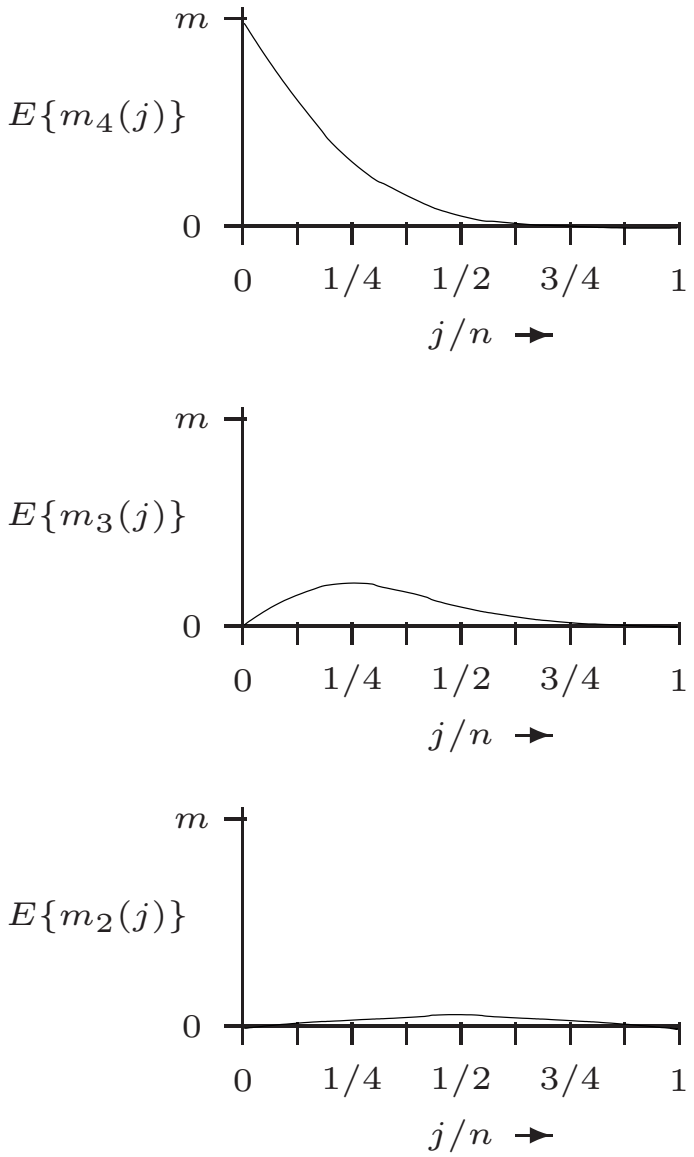
```
Let ψ be a CNF Boolean expression
Set M = ∅
Repeat the following:
   Choose an unassigned literal l in ψ
   If l is a positive literal, set M = M ∪ {l}
   Remove all clauses containing l from ψ
   Remove all literals l̄ from ψ
   If ψ is empty then return M
   If some clause in ψ is falsified return "give up"
```

When does this not give up with probability tending to 1?

Answer depends on the way literals are chosen

Flow Analysis





Flow Analysis

Example:

Unit clause heuristic: When there is a clause with one unassigned variable remaining, set the value of such a variable so as to satisfy its clause.

Flow Analysis

Example:

Unit clause heuristic: When there is a clause with one unassigned variable remaining, set the value of such a variable so as to satisfy its clause.

Intuitively: If the clause flow $w_1(j) < 1$ then any accumulation of unit clauses can be prevented and no clauses will ever be eliminated

Flow Analysis

Example:

Unit clause heuristic: When there is a clause with one unassigned variable remaining, set the value of such a variable so as to satisfy its clause.

Intuitively: If the clause flow $w_1(j) < 1$ then any accumulation of unit clauses can be prevented and no clauses will ever be eliminated

Analysis:

Write difference equations describing flows:

$$m_i(j+1) = m_i(j) + w_i(j) - w_{i-1}(j) - z_i(j), \quad \forall\, 0 \leq i \leq k, 1 < j < n$$

Take expectations and rearrange:

$$E\{m_i(j+1) - m_i(j)\} = E\{w_i(j)\} - E\{w_{i-1}(j)\} - E\{z_i(j)\}$$

Compute the expectations:

$$E\{z_i(j)\} = E\{E\{z_i(j)|m_i(j)\}\} = E\left\{\frac{i \cdot m_i(j)}{2(n-j)}\right\} = \frac{i \cdot E\{m_i(j)\}}{2(n-j)}$$

$$E\{w_i(j)\} = E\{E\{...\}\} = E\left\{\frac{(i+1)m_{i+1}(j)}{2(n-j)}\right\} = \frac{(i+1)E\{m_{i+1}(j)\}}{2(n-j)}$$

Compute the expectations:

$$E\{z_i(j)\} = E\{E\{z_i(j)|m_i(j)\}\} = E\left\{\frac{i \cdot m_i(j)}{2(n-j)}\right\} = \frac{i \cdot E\{m_i(j)\}}{2(n-j)}$$

$$E\{w_i(j)\} = E\{E\{...\}\} = E\left\{\frac{(i+1)m_{i+1}(j)}{2(n-j)}\right\} = \frac{(i+1)E\{m_{i+1}(j)\}}{2(n-j)}$$

Substitute into difference equations:

$$E\{m_i(j+1) - m_i(j)\} = \frac{(i+1)E\{m_{i+1}(j)\}}{2(n-j)} - \frac{i \cdot E\{m_i(j)\}}{n-j}, \quad i < k$$

$$E\{m_k(j+1) - m_k(j)\} = -\frac{k \cdot E\{m_k(j)\}}{n-j}$$

Compute the expectations:

$$E\{z_i(j)\} = E\{E\{z_i(j)|m_i(j)\}\} = E\left\{\frac{i \cdot m_i(j)}{2(n-j)}\right\} = \frac{i \cdot E\{m_i(j)\}}{2(n-j)}$$

$$E\{w_i(j)\} = E\{E\{...\}\} = E\left\{\frac{(i+1)m_{i+1}(j)}{2(n-j)}\right\} = \frac{(i+1)E\{m_{i+1}(j)\}}{2(n-j)}$$

Substitute into difference equations:

$$E\{m_i(j+1) - m_i(j)\} = \frac{(i+1)E\{m_{i+1}(j)\}}{2(n-j)} - \frac{i \cdot E\{m_i(j)\}}{n-j}, \quad i < k$$

$$E\{m_k(j+1) - m_k(j)\} = -\frac{k \cdot E\{m_k(j)\}}{n-j}$$

Switch to differential equations (use x for j/n):

$$\frac{d\bar{m}_i(x)}{dx} = \frac{(i+1)\bar{m}_{i+1}(x)}{2n(1-x)} - \frac{i \cdot \bar{m}_i(x)}{n-j}; \quad \bar{m}_k(0) = m/n, \ \bar{m}_i(0) = 0 \text{ for } i < k$$

Compute the expectations:

$$E\{z_i(j)\} = E\{E\{z_i(j)|m_i(j)\}\} = E\left\{\frac{i \cdot m_i(j)}{2(n-j)}\right\} = \frac{i \cdot E\{m_i(j)\}}{2(n-j)}$$

$$E\{w_i(j)\} = E\{E\{...\}\} = E\left\{\frac{(i+1)m_{i+1}(j)}{2(n-j)}\right\} = \frac{(i+1)E\{m_{i+1}(j)\}}{2(n-j)}$$

Substitute into difference equations:

$$E\{m_i(j+1) - m_i(j)\} = \frac{(i+1)E\{m_{i+1}(j)\}}{2(n-j)} - \frac{i \cdot E\{m_i(j)\}}{n-j}, \quad i < k$$

$$E\{m_k(j+1) - m_k(j)\} = -\frac{k \cdot E\{m_k(j)\}}{n-j}$$

Switch to differential equations (use x for j/n):

$$\frac{d\bar{m}_i(x)}{dx} = \frac{(i+1)\bar{m}_{i+1}(x)}{2n(1-x)} - \frac{i \cdot \bar{m}_i(x)}{n-j}; \quad \bar{m}_k(0) = m/n, \ \bar{m}_i(0) = 0 \text{ for } i < k$$

Solve:

$$\bar{m}_i(x) = \frac{1}{2^{k-i}}\binom{m}{n}\binom{k}{i}(1-x)^i x^{k-i}$$

Translation:

$$E\{m_i(j)\} = \frac{m}{2^{k-i}}\binom{k}{i}\left(1-\frac{j}{n}\right)^i\left(\frac{j}{n}\right)^{k-i}$$

Flow Analysis

The important flow:

$$E\{w_1(j)\} = \frac{E\{m_2(j)\}}{(n-j)} = \frac{1}{2^{k-2}} \binom{k}{2} \left(1 - \frac{j}{n}\right)^2 \left(\frac{j}{n}\right)^{k-2} m$$

Find location of maximum (set derivative = 0):

$$j^* = \left(\frac{k-2}{k-1}\right) n$$

So,

$E\{w_1(j^*)\} < 1$ when $\frac{m}{n} < \frac{2^{k-1}}{k} \left(\frac{k-1}{k-2}\right)^{k-1}$

for $k = 3$ this is $\frac{m}{n} < \frac{8}{3}$

Flow Analysis

The important flow:

$$E\{w_1(j)\} = \frac{E\{m_2(j)\}}{(n-j)} = \frac{1}{2^{k-2}} \binom{k}{2} \left(1 - \frac{j}{n}\right)^2 \left(\frac{j}{n}\right)^{k-2} m$$

Find location of maximum (set derivative = 0):

$$j^* = \left(\frac{k-2}{k-1}\right) n$$

So,

$$E\{w_1(j^*)\} < 1 \quad \text{when} \quad \frac{m}{n} < \frac{2^{k-1}}{k} \left(\frac{k-1}{k-2}\right)^{k-1}$$

for $k = 3$ this is $\frac{m}{n} < \frac{8}{3}$

Conclusion: unit clause heuristic succeeds with probability bounded by a constant when $\frac{m}{n} < \frac{8}{3}$.

Flow Analysis

What makes this work?

The clausal distribution is the same, up to parameters m and n, at each level (algorithm is myopic - no information is revealed)

There is pretty much never a sudden spurious flow

$$Pr(||C_i(j+1)| - |C_i(j)|| > n^{0.2}) = o(n^{-3})$$

The average flow change is pretty smooth

$$E\{|C_i(j+1)| - |C_i(j)|\} = f_i(j/n, |C_0(j)|/n, ..., |C_k(j)|/n) + o(1),$$

f_i is continuous and

$$|f_i(u_1, ..., u_{k+2}) - f_i(v_1, ..., v_{k+2})| \leq L \sum_{1 \leq i \leq j} |u_i - v_i|$$

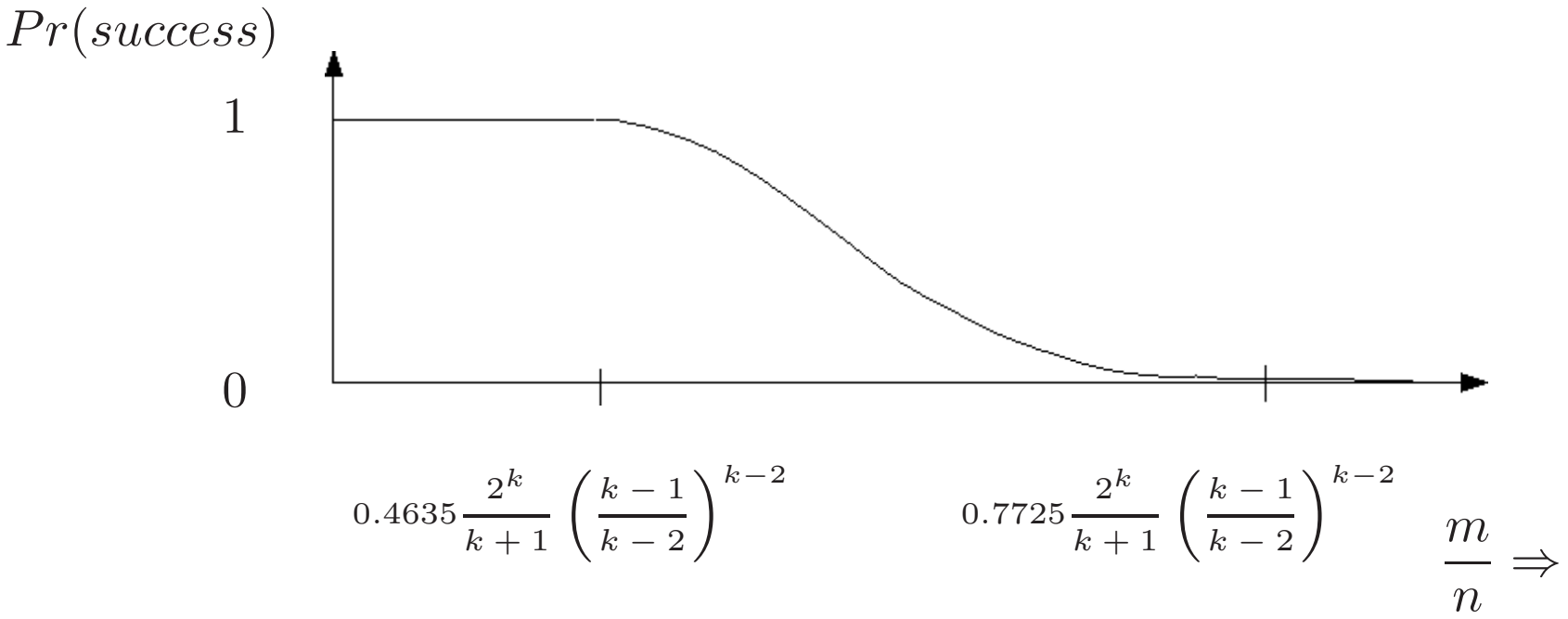
Flow Analysis

	good when $m/n <$	
literal selection	**k-SAT**	**3-SAT**
Choose from unit clause, otherwise randomly	$2^k/k$	2.66
Choose var with maximum difference between occurrences of positive and negative lits. Set value to maximize satisfied clauses	$c \cdot 2^k/k$	3.003
Choose randomly from a clause with fewest non-falsified literals	$1.125 \cdot 2^k/k$	3.09
Best possible myopic algorithm	$c \cdot 2^k/k$	3.26

literal selection, non-myopic algorithms

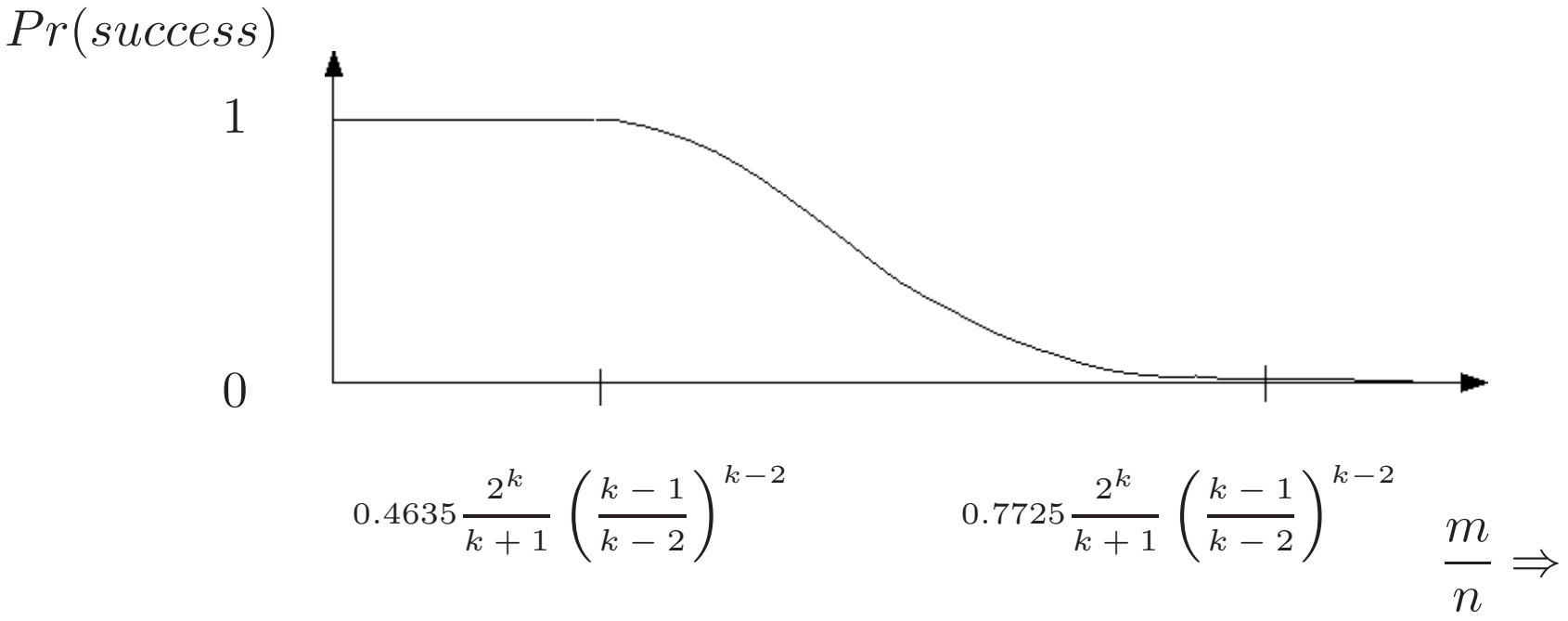
Greedy algorithm: maximize number of clauses satisfied, eliminate unit clauses when they exist	$c \cdot 2^k/k$?	3.52
Maximize the expected number of models of the reduced instance	$c \cdot 2^k/k$?	> 3.6?

Flow Analysis



Typical probability curve of these algorithms

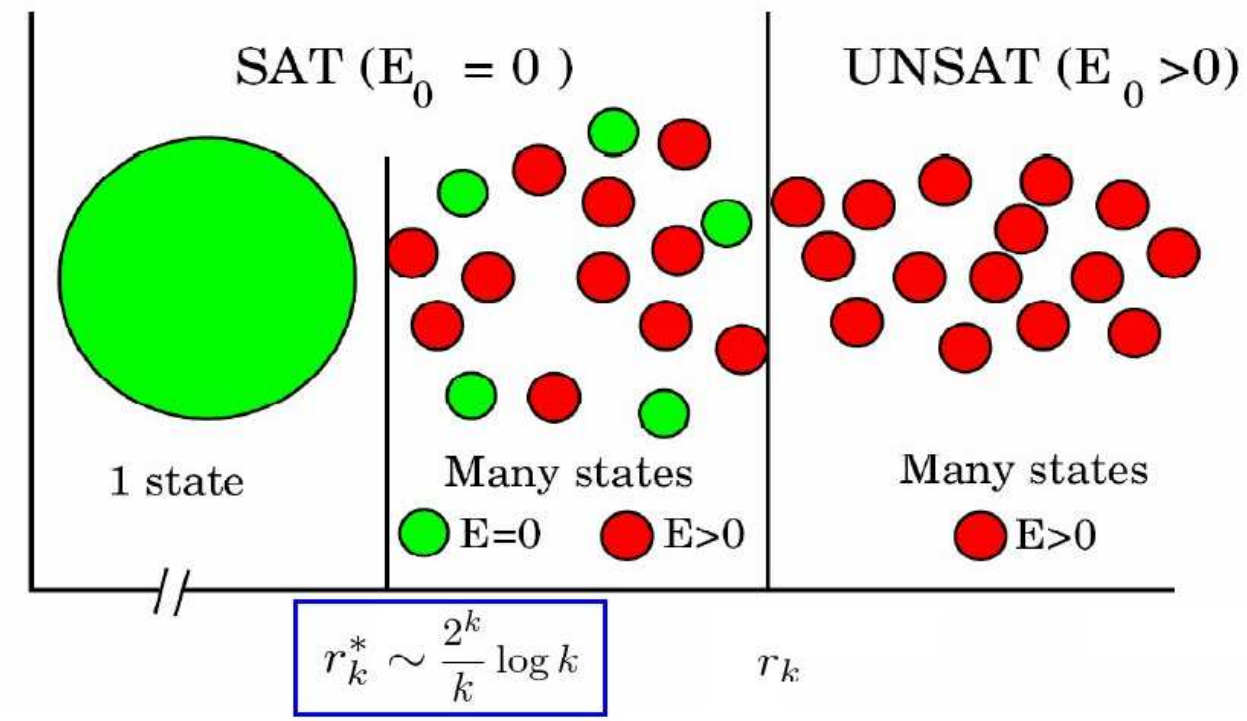
Flow Analysis



Typical probability curve of these algorithms

Is $\frac{2^k}{k}$ the crossover to unsatisfiability or is it 2^k?

An explanation?



Put points above a "floor" at distance = # clauses falsified
Create clusters of points around a lowest point

At $m/n = r_k^* \approx (2^k/k)\log(k)$:

Suddenly very many exponentially small clusters appear far apart from each other, surrounded by tall "mountains," and containing mostly frozen variables

Probabilistic Toolbox

Second Moment Method: show $Pr(P) \to 1$ as $n \to \infty$

Let P be some property of a formula

A witness for P: a structure whose presence in ψ implies P

Let $W = \{w : w$ is a witness for P in $\psi\}$

Let $X_i = \begin{cases} 1 & \text{if structure } s_i \in W \\ 0 & \text{otherwise} \end{cases}$

Let $X = \sum_i X_i$ and $E\{X_i\} = q$, then $E\{X\} \triangleq \mu = q|W|$

Suppose $var(X) \triangleq \sigma^2 = o(\mu^2)$. Then, since

$$Pr(X = 0) \leq \frac{\sigma^2}{\mu^2}$$

we get

$Pr(P) \to 1$ as $n \to \infty$

To show $\sigma^2 = o(\mu^2)$:

Start with one witness w chosen arbitrarily

Let A_w be all witnesses sharing at least one clause with w

Let D_w be all witnesses sharing no clause with w. Then

$$\sigma^2 = \mu(1-q) + \mu\left(\sum_{z \in A_w}(Pr(z|w) - q) + \sum_{z \in D_w}(Pr(z|w) - q)\right)$$

Need $|A_w| \ll |D_w|$ or "little" overlap among witnesses since

$Pr(z|w) = Pr(z) = q$ if $z \in D_w$ so $\sum_{z \in D_w}(Pr(z|w) - q) = 0$.

If $\mu \to \infty$ and $\sum_{z \in A_w} Pr(z|W) \to o(\mu)$ as $n \to \infty$ then

$Pr(P) \to 1$ as $n \to \infty$

Where is the Crossover?

Cannot apply the second moment method directly to k-SAT

- variance of the number of models is too high
  the reason: the asymmetry of k-SAT

Where is the Crossover?

Cannot apply the second moment method directly to k-SAT

- variance of the number of models is too high
  the reason: the asymmetry of k-SAT

Let z and w be assignments agreeing in αn variables

$$Pr(z \text{ satisfies } \psi \mid w \text{ satisfies } \psi) = \left(1 - \frac{1-\alpha^k}{2^k - 1}\right)^m$$

Variance gets too big, maximum occurs at $\alpha \neq 1/2$

$$\sum_{z \in A_w} Pr(z|w) = \sum_{0<\alpha\leq 1} \binom{n}{\alpha n} \left(1 - \frac{1-\alpha^k}{2^k - 1}\right)^m$$

Where is the Crossover?

Analyze a different problem: Not All Equal k-SAT

A NAE model: one for which every clause has at least one true and at least one false literal

$Pr(\exists$ a model for $\psi) > Pr(\exists$ a NAE-model for $\psi)$

Let $X =$ number of models, $X_{NAE} =$ number of NAE models

$Pr(X_{NAE} = 0) > Pr(X = 0)$

Where is the Crossover?

Analyze a different problem: Not All Equal k-SAT

A NAE model: one for which every clause has at least one true and at least one false literal

$Pr(\exists \text{ a model for } \psi) > Pr(\exists \text{ a NAE-model for } \psi)$

Let $X =$ number of models, $X_{NAE} =$ number of NAE models

$Pr(X_{NAE} = 0) > Pr(X = 0)$

$$Pr(z \text{ NAE-satisfies } \psi \mid w \text{ NAE-satisfies } \psi) = \left(1 - \frac{1 - \alpha^k - (1-\alpha)^k}{2^{k-1} - 1}\right)^m$$

Variance is low, maximum term occurs at $\alpha = 1/2$.

$$\sum_{0<\alpha\leq 1} \binom{n}{\alpha n} \left(1 - \frac{1 - \alpha^k - (1-\alpha)^k}{2^{k-1} - 1}\right)^m \approx \frac{2^n (1 - 2^{1-k})^m}{\sqrt{n}} \approx o(\mu)$$

$$Pr(X_{\text{NAE}} = 0) < \frac{1}{\mu_{\text{NAE}}} \approx \frac{1}{2^n (1 - 2^{k-1})^m} \quad \to 0 \text{ if } \quad \frac{m}{n} > \frac{1}{\lg(1 - 2^{k-1})} \approx 2^{k-1}$$

What About Easy Classes?

Examples:

Horn, Hidden Horn, 2-SAT, Extended Horn, q-Horn,
CC-balanced, SLUR, Matched, Linear Autarkies

Horn:

every clause has at most one positive literal
solved in linear time by unit clause algorithm

What About Easy Classes?

Examples:

Horn, Hidden Horn, 2-SAT, Extended Horn, q-Horn, CC-balanced, SLUR, Matched, Linear Autarkies

Horn:

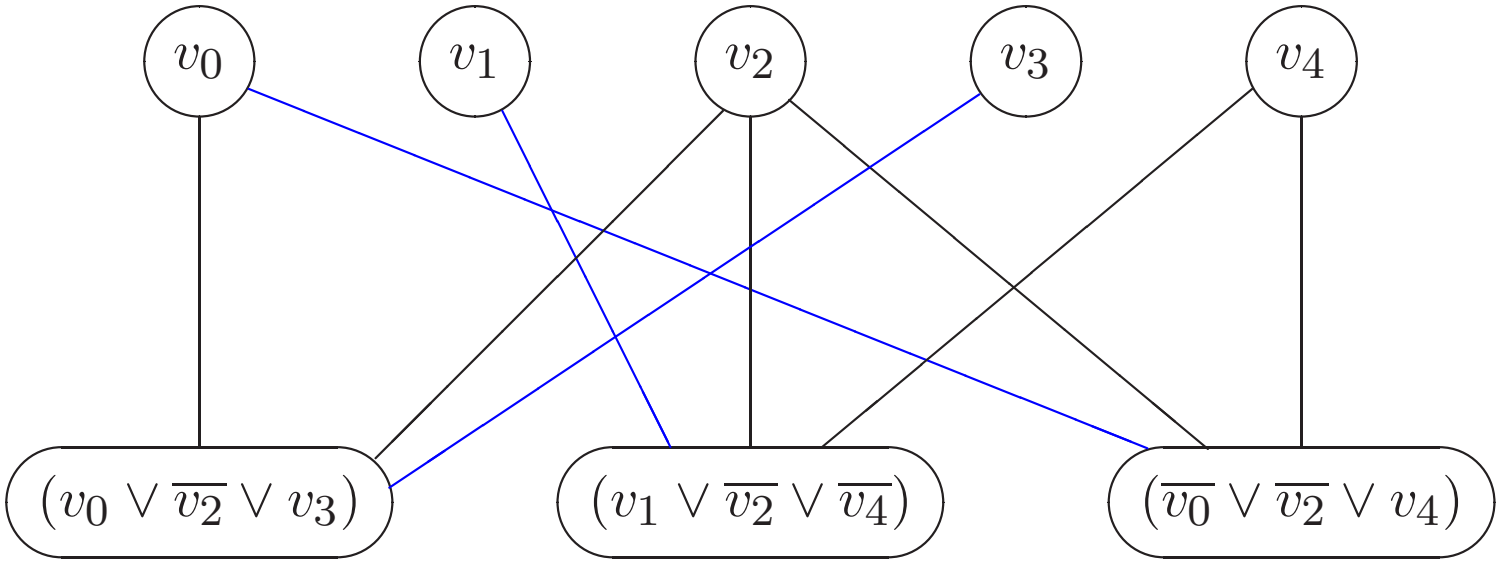
every clause has at most one positive literal
solved in linear time by unit clause algorithm

Probabilistic analysis of polytime solvable classes can reveal:

- What critically distinguishes an easy class from more difficult classes
- Whether one class is much larger than another incomparable class in a probabilistic sense

Polynomial Time Solvable Classes

Example: Matched



$$(v_0 \vee \overline{v_2} \vee v_3) \wedge (v_1 \vee \overline{v_2} \vee \overline{v_4}) \wedge (\overline{v_0} \vee \overline{v_2} \vee v_4)$$

Polynomial Time Solvable Classes

Example: Horn

$$(v_0 \vee \overline{v_1} \vee \overline{v_2}) \wedge (\overline{v_0}) \wedge (v_1 \vee \overline{v_3} \vee \overline{v_5}) \wedge (v_2) \wedge (\overline{v_2} \vee v_4)$$

Horn Solver (ψ)

Set $M \leftarrow \emptyset$.
Repeat the following until ψ has no positive unit clauses:
Choose variable v from a positive unit clause in ψ.
Set $M \leftarrow M \cup \{v\}$.
Remove clauses containing literal v from ψ.
Remove all literals $\overline{v}$ from ψ.
If ψ has an empty clause, output 'unsatisfiable."
Output M.

Unique minimum satisfying assignment w.r.t. 1

All assignments satisfying above include $v_2 = v_4 = 1$

Polynomial Time Solvable Classes

Example: q-Horn

variables

clauses

Horn	-1	0	1	0	0	0	0	0	0	0	0	Zero
	1	-1	-1	0	1	0	0	0	0	0	0	
	0	0	-1	-1	0	0	0	0	0	0	0	
	0	1	-1	0	0	0	0	0	0	0	0	
non-positive	0	-1	-1	0	0	0	0	1	0	1	0	2-SAT
	-1	0	0	-1	0	1	0	0	-1	0	0	
	0	0	0	-1	0	0	0	0	0	-1	1	
	0	0	-1	-1	0	0	-1	0	0	0	0	

Polynomial Time Solvable Classes

Example: q-Horn

variables

clauses											
Horn	-1	0	1	0	0	0	0	0	0	0	0	Zero
	1	-1	-1	0	1	0	0	0	0	0	0	
	0	0	-1	-1	0	0	0	0	0	0	0	
	0	1	-1	0	0	0	0	0	0	0	0	
non-positive	0	-1	-1	0	0	0	0	1	0	1	0	2-SAT
	-1	0	0	-1	0	1	0	0	-1	0	0	
	0	0	0	-1	0	0	0	0	0	-1	1	
	0	0	-1	-1	0	0	-1	0	0	0	0	

If the satisfiability index of a given formula is no greater than 1, then the formula is q-Horn.

The class of formulas with a satisfiability index greater than $1 + n^{-\epsilon}$, for any ϵ, is NP-complete.

Polynomial Time Solvable Classes

Vulnerability of q-Horn to particular cycles

$$\ldots (u_2 \vee \overline{u_1} \vee \ldots) \overset{u_1}{\rule{3em}{0.4pt}} (u_1 \vee \overline{v_0} \vee \overline{u_{3p}} \vee \ldots) \overset{u_{3p}}{\rule{3em}{0.4pt}} (u_{3p} \vee \overline{u_{3p-1}} \vee \ldots) \ \ldots$$

$$\Big| \qquad\qquad \Big| \, v_0 \qquad\qquad \Big|$$

$$\ldots (\overline{u_{p-1}} \vee u_p \vee \ldots) \underset{u_p}{\rule{3em}{0.4pt}} (\overline{u_p} \vee \overline{v_0} \vee u_{p+1} \vee \ldots) \underset{u_{p+1}}{\rule{3em}{0.4pt}} (\overline{u_{p+1}} \vee u_{p+2} \vee \ldots) \ldots$$

Polynomial Time Solvable Classes

Vulnerability of q-Horn to particular cycles

$$\ldots (u_2 \vee \overline{u_1} \vee \ldots) \xrightarrow{u_1} (u_1 \vee \overline{v_0} \vee \overline{u_{3p}} \vee \ldots) \xrightarrow{u_{3p}} (u_{3p} \vee \overline{u_{3p-1}} \vee \ldots) \ldots$$

$$\Big| \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \Big| \; v_0$$

$$\ldots (\overline{u_{p-1}} \vee u_p \vee \ldots) \xrightarrow[u_p]{} (\overline{u_p} \vee \overline{v_0} \vee u_{p+1} \vee \ldots) \xrightarrow[u_{p+1}]{} (\overline{u_{p+1}} \vee u_{p+2} \vee \ldots) \ldots$$

	Pr(random formula is of specified class)		
class	**as** $n \to \infty$	**k-SAT**	**3-SAT**
Horn	0	$m > \epsilon$	$m > \epsilon$
Hidden Horn	0	$m/n > 1/(k - \lg(k+1))$	$m/n > 1$
q-Horn	0	$m/n > 4/(k^2 - k)$	$m/n > 0.66$
SLUR	0	$m/n > 4/(k^2 - k)$	$m/n > 0.66$
Matched	1	$m/n < c_k,\ c_k \to 1$	$m/n < 0.64$
No Cycles	1	$m/n < 1.36/(k^2 - k)$	$m/n < 0.226$

Algorithms for Unsatisfiability

Resolution performs badly on random unsatisfiable formulas

$$Pr(\text{random } k\text{-SAT formula is unsatisfiable}) \rightarrow 1 \quad \text{if} \quad \frac{m}{n} > 2^k$$

$$Pr(\text{resolution does well}) \rightarrow 1 \quad \text{only if} \quad \frac{m}{n} > \left(\frac{n}{\lg(n)}\right)^{k-2}$$

Are there better alternatives?

Must avoid getting stuck on "sparse" nature of formulas

Algorithms for Unsatisfiability

Resolution performs badly on random unsatisfiable formulas

$$Pr(\text{random } k\text{-SAT formula is unsatisfiable}) \rightarrow 1 \quad \text{if} \quad \frac{m}{n} > 2^k$$

$$Pr(\text{resolution does well}) \rightarrow 1 \quad \text{only if} \quad \frac{m}{n} > \left(\frac{n}{\lg(n)}\right)^{k-2}$$

Are there better alternatives?

Must avoid getting stuck on "sparse" nature of formulas

One possibility: Hitting Set

Focus attention on clauses which are all positive or negative

Let $n^+ =$ min number of 1 valued variables to satisfy positives

Let $n^- =$ min number of 0 valued variables to satisfy negatives

If $n^+ + n^- > n$ then some variable must be set to 1 AND 0

That is impossible, conclude the formula is unsatisfiable

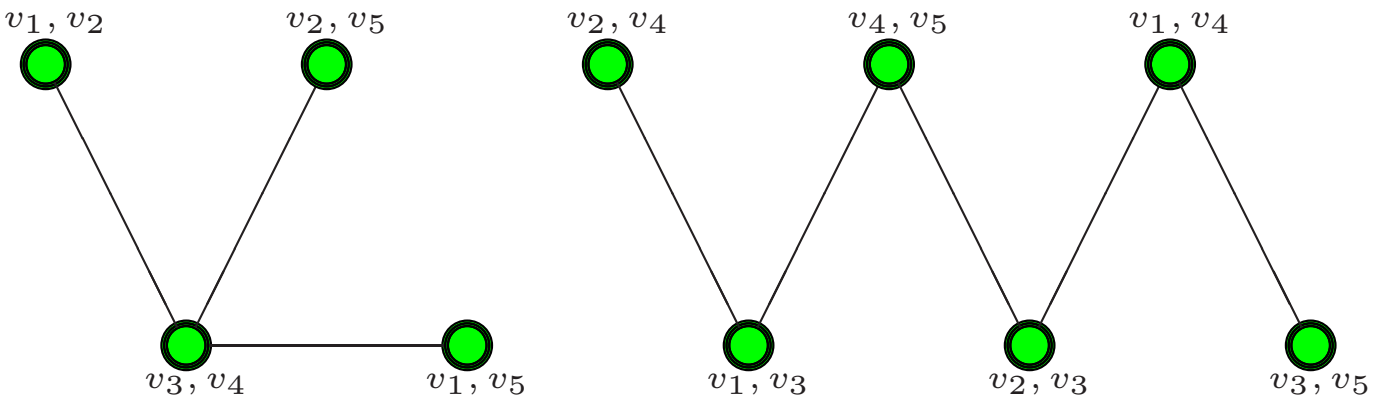
Hitting Set

Construct graphs G^+, G^-

Vertices are labeled as pairs of variables

Edge $\langle a, b \rangle \Leftrightarrow$ some clause contains all variables labeling a, b

$$(v_1 \vee v_2 \vee v_3 \vee v_4) \wedge (v_2 \vee v_3 \vee v_4 \vee v_5) \wedge (v_1 \vee v_3 \vee v_4 \vee v_5)$$



For $k = 4$, if a model exists, one graph has $|IS| > \frac{n^2}{8}$

Because, if ψ has a model, then there is some subset V' of $n/2$ variables s.t. ψ has no positive clause or negative clause taken from V'.

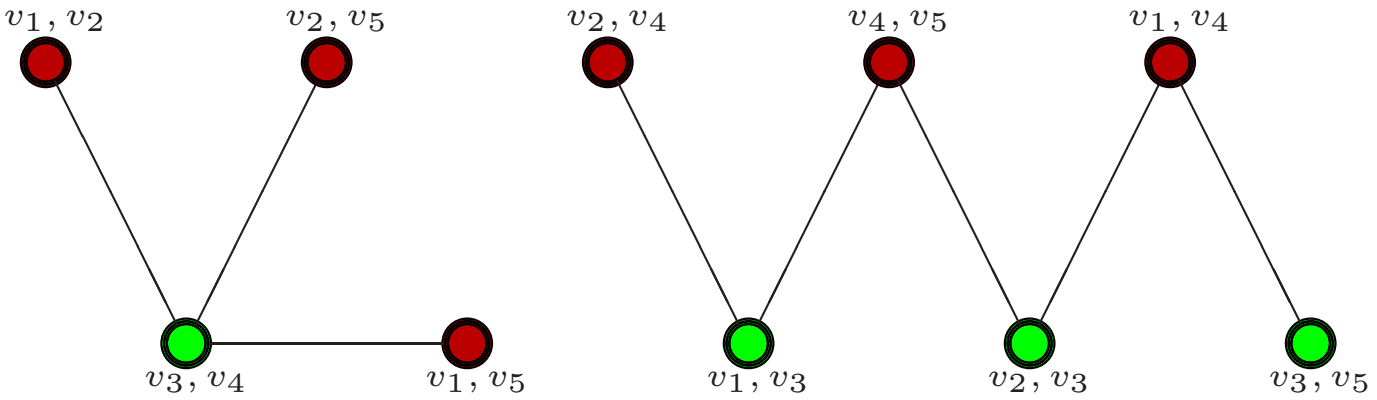
Hitting Set

Construct graphs G^+, G^-

Vertices are labeled as pairs of variables

Edge $\langle a, b \rangle \Leftrightarrow$ some clause contains all variables labeling a, b

$$(v_1 \vee v_2 \vee v_3 \vee v_4) \wedge (v_2 \vee v_3 \vee v_4 \vee v_5) \wedge (v_1 \vee v_3 \vee v_4 \vee v_5)$$



For $k = 4$, if a model exists, one graph has $|IS| > \frac{n^2}{8}$

Because, if ψ has a model, then there is some subset V' of $n/2$ variables s.t. ψ has no positive clause or negative clause taken from V'.

Hitting Set

Construct matrices M^+, M^-

Columns and rows are indexed on vertices

$$M_{i,j} = \begin{cases} -\frac{1-p}{p} & \text{if there is an edge between vertices } i \text{ and } j \\ 1 & \text{otherwise} \end{cases}$$

where p represents a probability that can be adjusted

Exploit relationship between max eigenvalue and max IS

$$\alpha(G^+) < \lambda_1(M^+) \quad \text{and} \quad \alpha(G^-) < \lambda_1(M^-)$$

Hitting Set

Construct matrices M^+, M^-

Columns and rows are indexed on vertices

$$M_{i,j} = \begin{cases} -\frac{1-p}{p} & \text{if there is an edge between vertices } i \text{ and } j \\ 1 & \text{otherwise} \end{cases}$$

where p represents a probability that can be adjusted

Exploit relationship between max eigenvalue and max IS

$$\alpha(G^+) < \lambda_1(M^+) \quad \text{and} \quad \alpha(G^-) < \lambda_1(M^-)$$

For purposes of proving a bound

If $p = \frac{\ln^7(n')}{n'}$ then $\max_i\{|\lambda_i(M)|\} = \frac{2n'}{\ln^{3.5}(n')}(1 + o(1))$ w.h.p.

This leads to $Pr(\text{HS does well}) \to 1$ if $\frac{m}{n} > n^{(k-2)/2}$

Recall $Pr(\text{resolution does well}) \to 1$ only if $\frac{m}{n} > n^{k-2}$.

A De-randomized Algorithm for MAXSAT

MAX k-SAT

Given: A CNF formula ψ with k literals per clause

Find: An assignment to the variables of ψ that satisfies a maximum number of its clauses.

Suppose ψ has variables $v_1, v_2, \ldots, v_n$. Define indicators

$$A^i_{t_1,\ldots,t_j} = \begin{cases} 1 & \text{if clause } i \text{ satisfied given } v_1 = t_1, \ldots, v_j = t_j \\ 0 & \text{otherwise} \end{cases}$$

What is the probability that $A^i_{t_1,\ldots,t_j} = 1$ for random $t_{j+1}, \ldots, t_n$?

for example: $Pr((\overline{v_1} \vee v_3 \vee \overline{v_5}) = 1 \mid t_1 = 1, t_2 = 0) = Pr(A^i_{1,0}) = 3/4$

A De-randomized Algorithm for MAXSAT

Let N be the number of satisfied clauses in ψ. Then

$$E\{N\} = \sum_{i=1}^{n} Pr(A^i) = (1 - 2^{-k})m$$

and

$$\begin{aligned} Pr(A^i_{t_1,\ldots,t_{j-1}}) &= (Pr(A^i_{t_1,\ldots,t_{j-1},1}) + Pr(A^i_{t_1,\ldots,t_{j-1},0}))/2 \\ &\leq \max\ \{Pr(A^i_{t_1,\ldots,t_j})\} \end{aligned}$$

so keep choosing a value t_j, for $j = 1, 2, \ldots$ that maximizes $Pr(A^i_{t_1,\ldots,t_j})$ to get

$$E\{N\} = (1 - 2^{-k})m = \sum_{i=1}^{n} Pr(A^i) \leq \sum_{i=1}^{n} \max\ \{Pr(A^i_{t_1})\} \ldots$$

$$\leq \sum_{i=1}^{n} \max\ \{Pr(A^i_{t_1,\ldots,t_n})\} = \#\ \text{clauses satisfied given}\ t_1, \ldots, t_n$$

A De-randomized Algorithm for MAXSAT

MAX k-SAT approximation algorithm uses this:

```
bool chooseValue (Variable *var) {
   double sum_pos=0.0, sum_neg=0.0, prob=0.5;
   for (int sz=1 ; sz <= k ; sz++) {
      sum_pos += var->no_clauses_as_pos_lit[sz]*prob;
      sum_neg += var->no_clauses_as_neg_lit[sz]*prob;
      prob *= 0.5;
   }
   return (sum_pos >= sum_neg) ? true : false;
}
```

and will always find an assignment that satisfies at least $(1 - 2^{-k})m$ clauses - for any input formula!