

# Automated Deduction

Laura Kovács

TU Wien

# Outline

## Satisfiability and Randomisation

Randomly Generated Clause Sets

Sharp Phase Transition

Randomised Algorithms for Satisfiability-Checking

Horn clauses

# Satisfiability and Randomisation

- ▶ SAT solving of **randomly generated set of clauses** are very hard for DPLL-based SAT solvers;
- ▶ Some **small randomly generated set of clauses** cannot be satisfied by DPLL-based SAT solvers, whereas **random SAT algorithms** can satisfy them;
- ▶ **Small randomly generated set of clauses** are useful for debugging SAT solvers;
- ▶ Experiments show that randomly generated set of clauses become from satisfiable to unsatisfiable in a **very narrow region**.

# Random Clause Generation

How can one generate a **random clause**?

# Random Clause Generation

How can one generate a random clause?  
Let's first generate a **random literal**.

# Random Clause Generation

How can one generate a random clause?  
Let's first generate a random literal.

- ▶ Fix a **number  $n$  of boolean variables**;

# Random Clause Generation

How can one generate a random clause?  
Let's first generate a random literal.

- ▶ Fix a number  $n$  of boolean variables;
- ▶ Select a literal among  $p_1, \dots, p_n, \neg p_1, \dots, \neg p_n$  with an equal probability.

# Random Clause Generation

How can one generate a random clause?

Let's first generate a random literal.

A **random clause** is a collection of random literals.

- ▶ Fix a number  $n$  of boolean variables;
- ▶ Select a literal among  $p_1, \dots, p_n, \neg p_1, \dots, \neg p_n$  with an equal probability.



# Random Clause Generation

How can one generate a random clause?

Let's first generate a random literal.

A random clause is a collection of random literals.

- ▶ Fix a number  $n$  of boolean variables;
- ▶ Select a literal among  $p_1, \dots, p_n, \neg p_1, \dots, \neg p_n$  with an equal probability.
- ▶ Fix the length  $k$  of the clause;

# Random Clause Generation

How can one generate a random clause?

Let's first generate a random literal.

A random clause is a collection of random literals.

- ▶ Fix a number  $n$  of boolean variables;
- ▶ Select a literal among  $p_1, \dots, p_n, \neg p_1, \dots, \neg p_n$  with an equal probability.
- ▶ Fix the length  $k$  of the clause;

Suppose we generate random clauses one after one. How does the set of models of this set change?

# SAT and $k$ -SAT

**SAT** is the problem of satisfiability checking for sets of clauses.

**$k$ -SAT** is the problem of satisfiability checking for sets of clauses of length  $k$ .

# SAT and $k$ -SAT

**SAT** is the problem of satisfiability checking for sets of clauses.

**$k$ -SAT** is the problem of satisfiability checking for sets of clauses of length  $k$ .

- ▶ SAT is **NP-complete**;

# SAT and $k$ -SAT

**SAT** is the problem of satisfiability checking for sets of clauses.

**$k$ -SAT** is the problem of satisfiability checking for sets of clauses of length  $k$ .

- ▶ SAT is **NP-complete**;
- ▶ **2-SAT** is decidable in linear time;

# SAT and $k$ -SAT

**SAT** is the problem of satisfiability checking for sets of clauses.

**$k$ -SAT** is the problem of satisfiability checking for sets of clauses of length  $k$ .

- ▶ SAT is **NP-complete**;
- ▶ **2-SAT** is decidable in linear time;
- ▶ **3-SAT** is NP-complete.

# SAT and $k$ -SAT

**SAT** is the problem of satisfiability checking for sets of clauses.

**$k$ -SAT** is the problem of satisfiability checking for sets of clauses of length  $k$ .

- ▶ SAT is **NP-complete**;
- ▶ **2-SAT** is decidable in linear time;
- ▶ **3-SAT** is NP-complete.

There is a simple reduction of SAT to **3-SAT** based on the same ideas as used for generating short clausal forms (naming). Take a clause having more than 3 literals:

$$L_1 \vee L_2 \vee L_3 \vee L_4 \dots$$

And replace it by two clauses:

$$\begin{aligned} L_1 \vee L_2 \vee n \\ \neg n \vee L_3 \vee L_4 \dots \end{aligned}$$

where  $n$  is a new variable.

Example (obtained by a program) for  $n = 5$  and  $k = 2$

$p_1$	$p_2$	$p_3$	$p_4$	$p_5$	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$
0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	1	0	0	0	1
0	0	0	1	0	1	0	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	0	0	1	0	1	0	0
0	0	1	0	1	1	0	1	0	1
0	0	1	1	0	1	0	1	1	0
0	0	1	1	1	1	0	1	1	1
0	1	0	0	0	1	1	0	0	0
0	1	0	0	1	1	1	0	0	1
0	1	0	1	0	1	1	0	1	0
0	1	0	1	1	1	1	0	1	1
0	1	1	0	0	1	1	1	0	0
0	1	1	0	1	1	1	1	0	1
0	1	1	1	0	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1

Number of models: 32



# Example (obtained by a program) for $n = 5$ and $k = 2$

$\neg p_2 \vee \neg p_3$

$p_1$	$p_2$	$p_3$	$p_4$	$p_5$
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1
0	0	1	0	0
0	0	1	0	1
0	0	1	1	0
0	0	1	1	1
0	1	0	0	0
0	1	0	0	1
0	1	0	1	0
0	1	0	1	1
0	1	1	0	0
0	1	1	0	1
0	1	1	1	0
0	1	1	1	1

$p_1$	$p_2$	$p_3$	$p_4$	$p_5$
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	0
1	0	1	0	1
1	0	1	1	0
1	0	1	1	1
1	1	0	0	0
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1
1	1	1	0	0
1	1	1	0	1
1	1	1	1	0
1	1	1	1	1

Number of models: 32

# Example (obtained by a program) for $n = 5$ and $k = 2$

	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$
$\neg p_2 \vee \neg p_3$	0	0	0	0	0	1	0	0	0	0
	0	0	0	0	1	1	0	0	0	1
	0	0	0	1	0	1	0	0	1	0
	0	0	0	1	1	1	0	0	1	1
	0	0	1	0	0	1	0	1	0	0
	0	0	1	0	1	1	0	1	0	1
	0	0	1	1	0	1	0	1	1	0
	0	0	1	1	1	1	0	1	1	1
	0	1	0	0	0	1	1	0	0	0
	0	1	0	0	1	1	1	0	0	1
	0	1	0	1	0	1	1	0	1	0
	0	1	0	1	1	1	1	0	1	1

Number of models: 24

# Example (obtained by a program) for $n = 5$ and $k = 2$

	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$
$\neg p_2 \vee \neg p_3$	0	0	0	0	0	1	0	0	0	0
$\neg p_2 \vee p_1$	0	0	0	0	1	1	0	0	0	1
	0	0	0	1	0	1	0	0	1	0
	0	0	0	1	1	1	0	0	1	1
	0	0	1	0	0	1	0	1	0	0
	0	0	1	0	1	1	0	1	0	1
	0	0	1	1	0	1	0	1	1	0
	0	0	1	1	1	1	0	1	1	1
	0	1	0	0	0	1	1	0	0	0
	0	1	0	0	1	1	1	0	0	1
	0	1	0	1	0	1	1	0	1	0
	0	1	0	1	1	1	1	0	1	1

Number of models: 24

# Example (obtained by a program) for $n = 5$ and $k = 2$

	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$
$\neg p_2 \vee \neg p_3$	0	0	0	0	0	1	0	0	0	0
$\neg p_2 \vee p_1$	0	0	0	0	1	1	0	0	0	1
	0	0	0	1	0	1	0	0	1	0
	0	0	0	1	1	1	0	0	1	1
	0	0	1	0	0	1	0	1	0	0
	0	0	1	0	1	1	0	1	0	1
	0	0	1	1	0	1	0	1	1	0
	0	0	1	1	1	1	0	1	1	1
						1	1	0	0	0
						1	1	0	0	1
						1	1	0	1	0
						1	1	0	1	1

Number of models: 20

# Example (obtained by a program) for $n = 5$ and $k = 2$

	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$
$\neg p_2 \vee \neg p_3$	0	0	0	0	0	1	0	0	0	0
$\neg p_2 \vee p_1$	0	0	0	0	1	1	0	0	0	1
$\neg p_2 \vee p_2$	0	0	0	1	0	1	0	0	1	0
	0	0	0	1	1	1	0	0	1	1
	0	0	1	0	0	1	0	1	0	0
	0	0	1	0	1	1	0	1	0	1
	0	0	1	1	0	1	0	1	1	0
	0	0	1	1	1	1	0	1	1	1
						1	1	0	0	0
						1	1	0	0	1
						1	1	0	1	0
						1	1	0	1	1

Number of models: 20

# Example (obtained by a program) for $n = 5$ and $k = 2$

	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$
$\neg p_2 \vee \neg p_3$	0	0	0	0	0	1	0	0	0	0
$\neg p_2 \vee p_1$	0	0	0	0	1	1	0	0	0	1
$\neg p_2 \vee p_2$	0	0	0	1	0	1	0	0	1	0
$p_1 \vee p_1$	0	0	0	1	1	1	0	0	1	1
	0	0	1	0	0	1	0	1	0	0
	0	0	1	0	1	1	0	1	0	1
	0	0	1	1	0	1	0	1	1	0
	0	0	1	1	1	1	0	1	1	1
						1	1	0	0	0
						1	1	0	0	1
						1	1	0	1	0
						1	1	0	1	1

Number of models: 20

# Example (obtained by a program) for $n = 5$ and $k = 2$

	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$
$\neg p_2 \vee \neg p_3$					
$\neg p_2 \vee p_1$					
$\neg p_2 \vee p_2$					
$p_1 \vee p_1$					

$p_1$	$p_2$	$p_3$	$p_4$	$p_5$
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	0
1	0	1	0	1
1	0	1	1	0
1	0	1	1	1
1	1	0	0	0
1	1	0	0	1
1	1	0	1	0
1	1	0	1	1

Number of models: 12

# Example (obtained by a program) for $n = 5$ and $k = 2$

	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$
$\neg p_2 \vee \neg p_3$					
$\neg p_2 \vee p_1$					
$\neg p_2 \vee p_2$					
$p_1 \vee p_1$					
$\neg p_5 \vee p_5$					

	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$
1	0	0	0	0	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	0	1	1
1	0	1	0	0	0
1	0	1	0	0	1
1	0	1	1	1	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	0	1	1	1

Number of models: 12



# Example (obtained by a program) for $n = 5$ and $k = 2$

	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$
$\neg p_2 \vee \neg p_3$					
$\neg p_2 \vee p_1$					
$\neg p_2 \vee p_2$					
$p_1 \vee p_1$					
$\neg p_5 \vee p_5$					
$p_4 \vee p_5$					

	$p_1$	$p_2$	$p_3$	$p_4$	$p_5$
	1	0	0	0	0
	1	0	0	0	1
	1	0	0	1	0
	1	0	0	1	1
	1	0	1	0	0
	1	0	1	0	1
	1	0	1	1	0
	1	0	1	1	1
	1	1	0	0	0
	1	1	0	0	1
	1	1	0	1	0
	1	1	0	1	1

Number of models: 12

# Example (obtained by a program) for $n = 5$ and $k = 2$

	<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>		<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>
$\neg p_2 \vee \neg p_3$						1	0	0	0	1	
$\neg p_2 \vee p_1$						1	0	0	1	0	
$\neg p_2 \vee p_2$						1	0	0	1	1	
$p_1 \vee p_1$											
$\neg p_5 \vee p_5$						1	0	1	0	1	
$p_4 \vee p_5$						1	0	1	1	0	
						1	0	1	1	1	
						1	1	0	0	1	
						1	1	0	1	0	
						1	1	0	1	1	

Number of models: 9

# Example (obtained by a program) for $n = 5$ and $k = 2$

	<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>		<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>
$\neg p_2 \vee \neg p_3$							1	0	0	0	1
$\neg p_2 \vee p_1$							1	0	0	1	0
$\neg p_2 \vee p_2$							1	0	0	1	1
$p_1 \vee p_1$											
$\neg p_5 \vee p_5$							1	0	1	0	1
$p_4 \vee p_5$							1	0	1	1	0
$\neg p_5 \vee \neg p_3$							1	0	1	1	1
							1	1	0	0	1
							1	1	0	1	0
							1	1	0	1	1

Number of models: 9

# Example (obtained by a program) for $n = 5$ and $k = 2$

	<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>		<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>
$\neg p_2 \vee \neg p_3$							1	0	0	0	1
$\neg p_2 \vee p_1$							1	0	0	1	0
$\neg p_2 \vee p_2$							1	0	0	1	1
$p_1 \vee p_1$											
$\neg p_5 \vee p_5$											
$p_4 \vee p_5$							1	0	1	1	0
$\neg p_5 \vee \neg p_3$											
							1	1	0	0	1
							1	1	0	1	0
							1	1	0	1	1

Number of models: 7

# Example (obtained by a program) for $n = 5$ and $k = 2$

	<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>		<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>
$\neg p_2 \vee \neg p_3$							1	0	0	0	1
$\neg p_2 \vee p_1$							1	0	0	1	0
$\neg p_2 \vee p_2$							1	0	0	1	1
$p_1 \vee p_1$											
$\neg p_5 \vee p_5$											
$p_4 \vee p_5$							1	0	1	1	0
$\neg p_5 \vee \neg p_3$											
$p_2 \vee \neg p_4$							1	1	0	0	1
							1	1	0	1	0
							1	1	0	1	1

Number of models: 7

# Example (obtained by a program) for $n = 5$ and $k = 2$

	<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>		<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>
$\neg p_2 \vee \neg p_3$							1	0	0	0	1
$\neg p_2 \vee p_1$											
$\neg p_2 \vee p_2$											
$p_1 \vee p_1$											
$\neg p_5 \vee p_5$											
$p_4 \vee p_5$											
$\neg p_5 \vee \neg p_3$											
$p_2 \vee \neg p_4$							1	1	0	0	1
							1	1	0	1	0
							1	1	0	1	1

Number of models: 4

# Example (obtained by a program) for $n = 5$ and $k = 2$

	<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>		<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>
$\neg p_2 \vee \neg p_3$							1	0	0	0	1
$\neg p_2 \vee p_1$											
$\neg p_2 \vee p_2$											
$p_1 \vee p_1$											
$\neg p_5 \vee p_5$											
$p_4 \vee p_5$											
$\neg p_5 \vee \neg p_3$											
$p_2 \vee \neg p_4$							1	1	0	0	1
$p_5 \vee \neg p_2$							1	1	0	1	1

Number of models: 4

# Example (obtained by a program) for $n = 5$ and $k = 2$

	<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>		<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>
$\neg p_2 \vee \neg p_3$							1	0	0	0	1
$\neg p_2 \vee p_1$											
$\neg p_2 \vee p_2$											
$p_1 \vee p_1$											
$\neg p_5 \vee p_5$											
$p_4 \vee p_5$											
$\neg p_5 \vee \neg p_3$											
$p_2 \vee \neg p_4$							1	1	0	0	1
$p_5 \vee \neg p_2$							1	1	0	1	1

Number of models: 3



# Example (obtained by a program) for $n = 5$ and $k = 2$

	<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>		<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>
$\neg p_2 \vee \neg p_3$							1	0	0	0	1
$\neg p_2 \vee p_1$											
$\neg p_2 \vee p_2$											
$p_1 \vee p_1$											
$\neg p_5 \vee p_5$											
$p_4 \vee p_5$											
$\neg p_5 \vee \neg p_3$											
$p_2 \vee \neg p_4$							1	1	0	0	1
$p_5 \vee \neg p_2$											
$p_5 \vee p_2$							1	1	0	1	1

Number of models: 3

# Example (obtained by a program) for $n = 5$ and $k = 2$

	<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>		<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>
$\neg p_2 \vee \neg p_3$							1	0	0	0	1
$\neg p_2 \vee p_1$											
$\neg p_2 \vee p_2$											
$p_1 \vee p_1$											
$\neg p_5 \vee p_5$											
$p_4 \vee p_5$											
$\neg p_5 \vee \neg p_3$											
$p_2 \vee \neg p_4$											
$p_5 \vee \neg p_2$											
$p_5 \vee p_2$											

Number of models: 1

# Example (obtained by a program) for $n = 5$ and $k = 2$

	<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>		<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>
$\neg p_2 \vee \neg p_3$							1	0	0	0	1
$\neg p_2 \vee p_1$											
$\neg p_2 \vee p_2$											
$p_1 \vee p_1$											
$\neg p_5 \vee p_5$											
$p_4 \vee p_5$											
$\neg p_5 \vee \neg p_3$											
$p_2 \vee \neg p_4$											
$p_5 \vee \neg p_2$											
$p_5 \vee p_2$											
$\neg p_1 \vee \neg p_4$											

Number of models: 1

# Example (obtained by a program) for $n = 5$ and $k = 2$

	<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>		<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>
$\neg p_2 \vee \neg p_3$											
$\neg p_2 \vee p_1$											
$\neg p_2 \vee p_2$											
$p_1 \vee p_1$											
$\neg p_5 \vee p_5$											
$p_4 \vee p_5$											
$\neg p_5 \vee \neg p_3$											
$p_2 \vee \neg p_4$											
$p_5 \vee \neg p_2$											
$p_5 \vee p_2$											
$\neg p_1 \vee \neg p_4$											
$p_5 \vee p_2$											

Number of models: 1

# Example (obtained by a program) for $n = 5$ and $k = 2$

	<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>		<u><math>p_1</math></u>	<u><math>p_2</math></u>	<u><math>p_3</math></u>	<u><math>p_4</math></u>	<u><math>p_5</math></u>
$\neg p_2 \vee \neg p_3$											
$\neg p_2 \vee p_1$											
$\neg p_2 \vee p_2$											
$p_1 \vee p_1$											
$\neg p_5 \vee p_5$											
$p_4 \vee p_5$											
$\neg p_5 \vee \neg p_3$											
$p_2 \vee \neg p_4$											
$p_5 \vee \neg p_2$											
$p_5 \vee p_2$											
$\neg p_1 \vee \neg p_4$											
$p_5 \vee p_2$											
$\neg p_1 \vee \neg p_5$											
							1	0	0	0	1

Number of models: 1

# Example (obtained by a program) for $n = 5$ and $k = 2$

$p_1$   $p_2$   $p_3$   $p_4$   $p_5$

$p_1$   $p_2$   $p_3$   $p_4$   $p_5$

$$\neg p_2 \vee \neg p_3$$

$$\neg p_2 \vee p_1$$

$$\neg p_2 \vee p_2$$

$$p_1 \vee p_1$$

$$\neg p_5 \vee p_5$$

$$p_4 \vee p_5$$

$$\neg p_5 \vee \neg p_3$$

$$p_2 \vee \neg p_4$$

$$p_5 \vee \neg p_2$$

$$p_5 \vee p_2$$

$$\neg p_1 \vee \neg p_4$$

$$p_5 \vee p_2$$

$$\neg p_1 \vee \neg p_5$$

Number of models: 0

This set of 13 clauses is unsatisfiable.

# Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

# Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

Fix:

- ▶ Number  $n$  of boolean variables;



# Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

Fix:

- ▶ Number  $n$  of boolean variables;
- ▶ Number  $k$  of **literals per clause**, so we will generate  $k$ -SAT instances;

# Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

Fix:

- ▶ Number  $n$  of boolean variables;
- ▶ Number  $k$  of **literals per clause**, so we will generate  $k$ -SAT instances;
- ▶ Number  $m$  of the **clauses**.

# Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

Fix:

- ▶ Number  $n$  of boolean variables;
- ▶ Number  $k$  of **literals per clause**, so we will generate  $k$ -SAT instances;
- ▶ Number  $m$  of the **clauses**.

Generate  $m$  clauses, each one has  $k$  literals **randomly generated** among  $p_1, \dots, p_n, \neg p_1, \dots, \neg p_n$  with an equal probability.

# Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

Fix:

- ▶ Number  $n$  of boolean variables;
- ▶ Number  $k$  of **literals per clause**, so we will generate  $k$ -SAT instances;
- ▶ Number  $m$  of the **clauses**.

Generate  $m$  clauses, each one has  $k$  literals **randomly generated** among  $p_1, \dots, p_n, \neg p_1, \dots, \neg p_n$  with an equal probability.

**Exercise:** What are the probabilities that the resulting sets are unsatisfiable for  $m = 1$  and  $m = 2$ ?

# Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

Fix:

- ▶ Number  $n$  of boolean variables;
- ▶ Number  $k$  of **literals per clause**, so we will generate  $k$ -SAT instances;
- ▶ Number  $m$  of the **clauses**.

Generate  $m$  clauses, each one has  $k$  literals **randomly generated** among  $p_1, \dots, p_n, \neg p_1, \dots, \neg p_n$  with an equal probability.

Note that the probability is a **monotone** function: the more clauses we generate, the higher chance we have that the set is unsatisfiable.

# Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

Fix:

- ▶ Number  $n$  of boolean variables;
- ▶ Number  $k$  of **literals per clause**, so we will generate  $k$ -SAT instances;
- ▶ Real number  $r$ : **ratio of clauses per variable**.

Generate  $[rn]$  clauses, each one has  $k$  literals **randomly generated** among  $p_1, \dots, p_n, \neg p_1, \dots, \neg p_n$  with an equal probability.

Note that the probability is a **monotone** function: the more clauses we generate, the higher chance we have that the set is unsatisfiable.

# Random Clause Generation

We are interested in the probability that a set of clauses of a given size is unsatisfiable.

Fix:

- ▶ Number  $n$  of boolean variables;
- ▶ Number  $k$  of **literals per clause**, so we will generate  $k$ -SAT instances;
- ▶ Real number  $r$ : **ratio of clauses per variable**.

Generate  $\lceil rn \rceil$  clauses, each one has  $k$  literals **randomly generated** among  $p_1, \dots, p_n, \neg p_1, \dots, \neg p_n$  with an equal probability.

Note that the probability is a **monotone** function: the more clauses we generate, the higher chance we have that the set is unsatisfiable.

Denote by  $\pi(r, n)$  the **probability** that a randomly generated set of  $\lceil rn \rceil$   $k$ -clauses is unsatisfiable.

# Roulette





# SAT Roulette



We will generate random instances of 2-SAT with 5-variables.

You will bet on whether the resulting set of clauses is satisfiable or not.

# SAT Roulette



We will generate random instances of 2-SAT with 5-variables.

You will bet on whether the resulting set of clauses is satisfiable or not.

- ▶ What will you bet on if we generate 5 clauses?

# SAT Roulette



We will generate random instances of 2-SAT with 5-variables.

You will bet on whether the resulting set of clauses is satisfiable or not.

- ▶ What will you bet on if we generate 5 clauses?
- ▶ What will you bet on if we generate 100 clauses?

# SAT Roulette



We will generate random instances of 2-SAT with 5-variables.

You will bet on whether the resulting set of clauses is satisfiable or not.

- ▶ What will you bet on if we generate 5 clauses?
- ▶ What will you bet on if we generate 100 clauses?
- ▶ What will you bet on if we generate 15 clauses?

# SAT Roulette

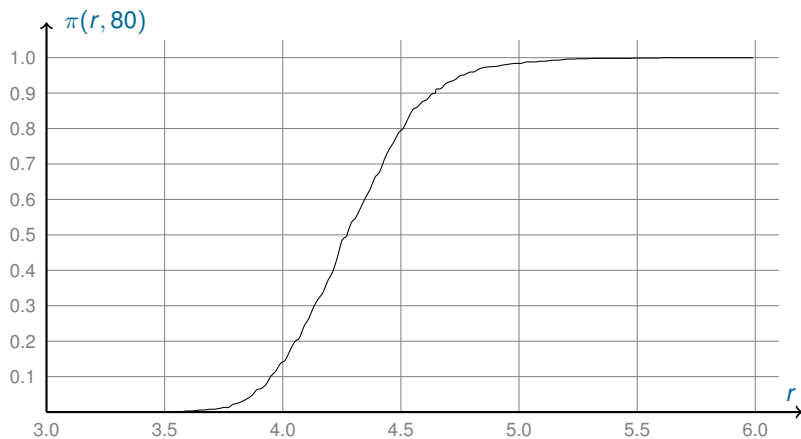


We will generate random instances of 3-SAT with 5-variables.

You will bet on whether the resulting set of clauses is satisfiable or not.

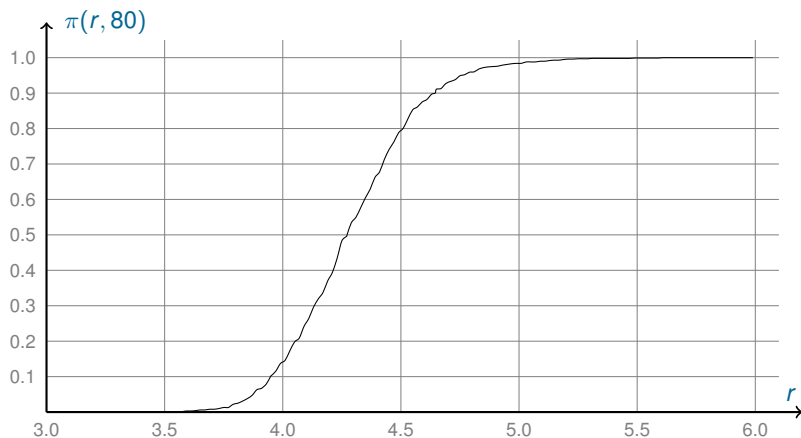
- ▶ What will you bet on if we generate 5 clauses?
- ▶ What will you bet on if we generate 100 clauses?
- ▶ What will you bet on if we generate 15 clauses?

# Probability of obtaining an unsatisfiable set of 3-SAT



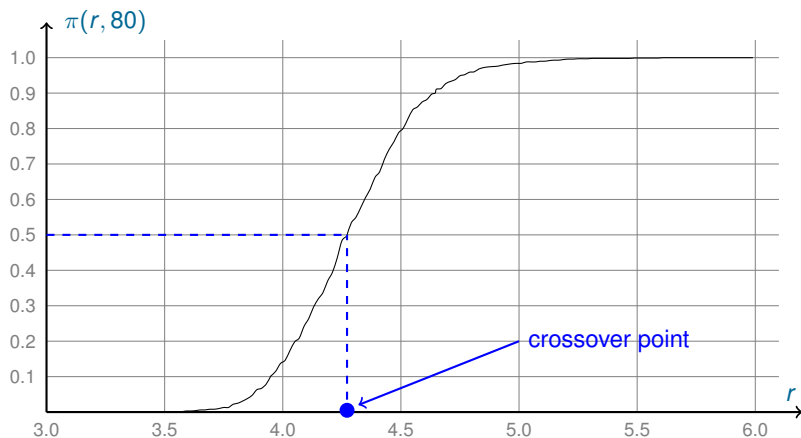
# Probability of obtaining an unsatisfiable set of 3-SAT

**Crossover point:** the value of  $r$  at which the probability crosses 0.5.



# Probability of obtaining an unsatisfiable set of 3-SAT

**Crossover point:** the value of  $r$  at which the probability crosses 0.5.





## $\epsilon$ -window

Take a (small) number  $0 < \epsilon < 0.5$ .  $\epsilon$ -window is the interval of values of  $r$  where the probability is between  $\epsilon$  and  $1 - \epsilon$ .

## $\epsilon$ -window

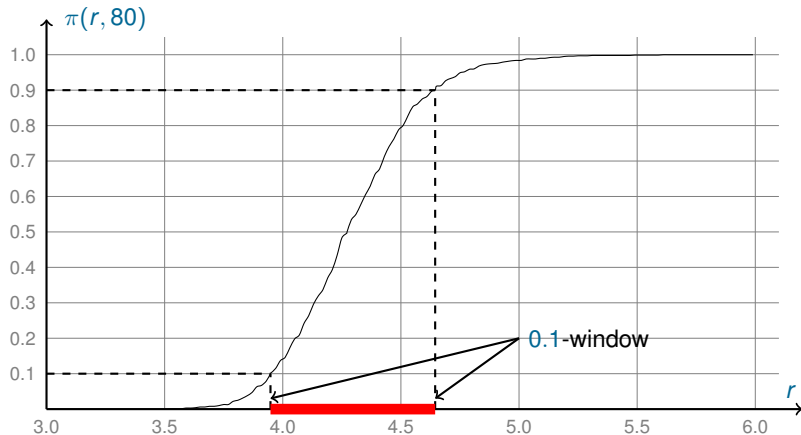
Take a (small) number  $0 < \epsilon < 0.5$ .  $\epsilon$ -window is the interval of values of  $r$  where the probability is between  $\epsilon$  and  $1 - \epsilon$ .

For example, take  $\epsilon = 0.1$ .

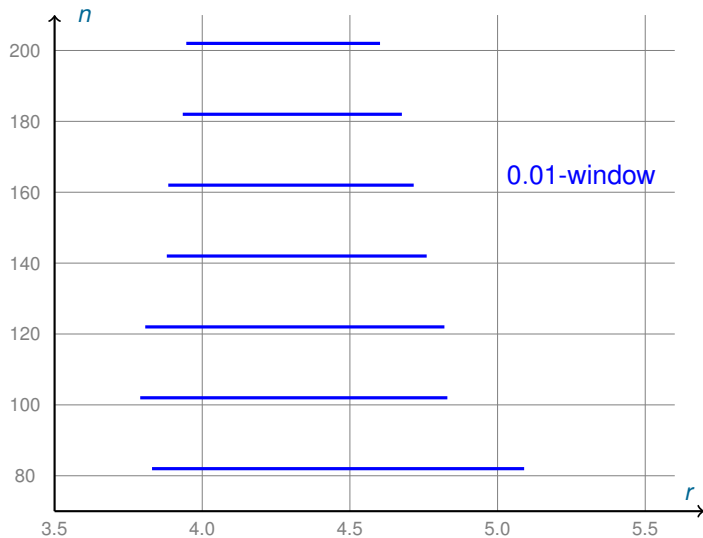
## $\epsilon$ -window

Take a (small) number  $0 < \epsilon < 0.5$ .  $\epsilon$ -window is the interval of values of  $r$  where the probability is between  $\epsilon$  and  $1 - \epsilon$ .

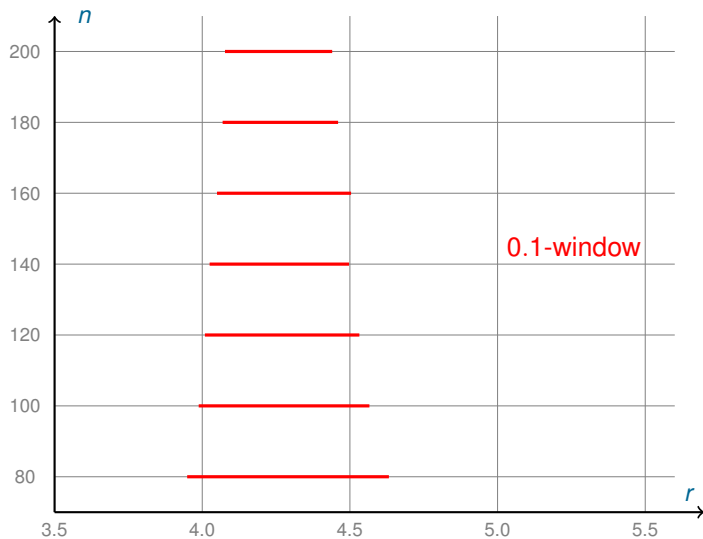
For example, take  $\epsilon = 0.1$ .



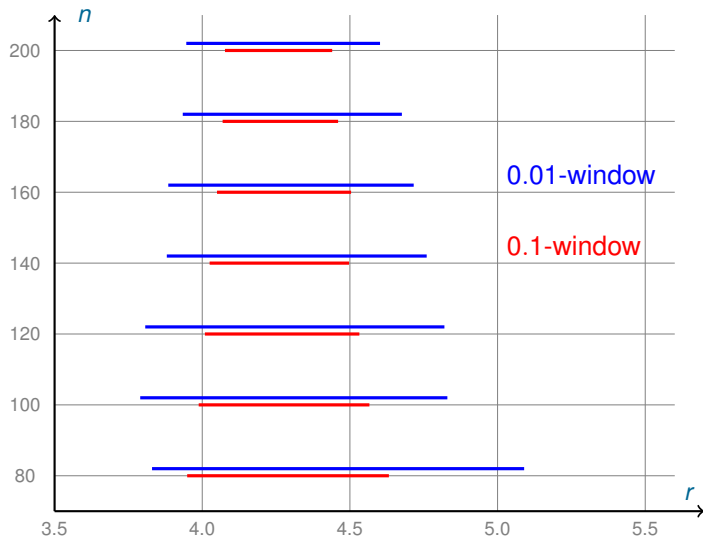
# Scaling Window Effect



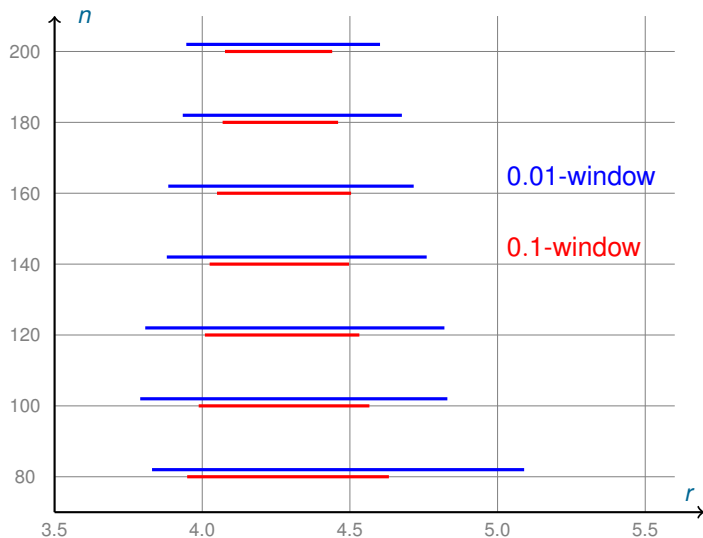
# Scaling Window Effect



# Scaling Window Effect

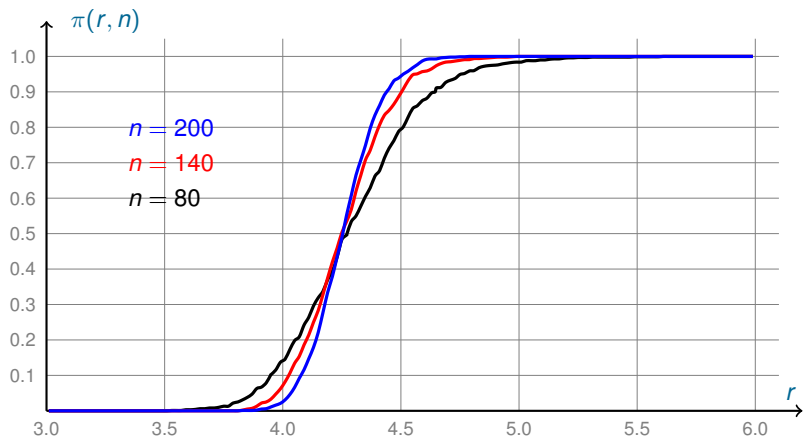


# Scaling Window Effect



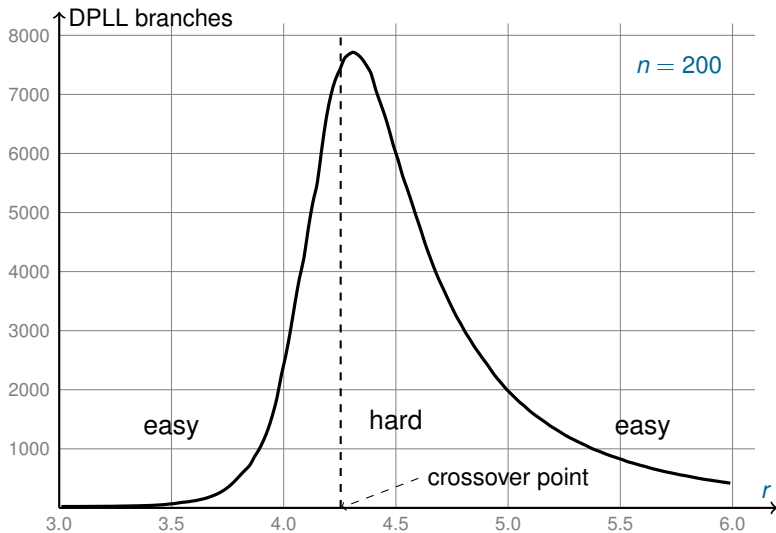
Conjecture: for  $n \rightarrow \infty$  every  $\epsilon$ -window “degenerates into a point”.

# Sharp Phase Transition of $\pi(r, n)$

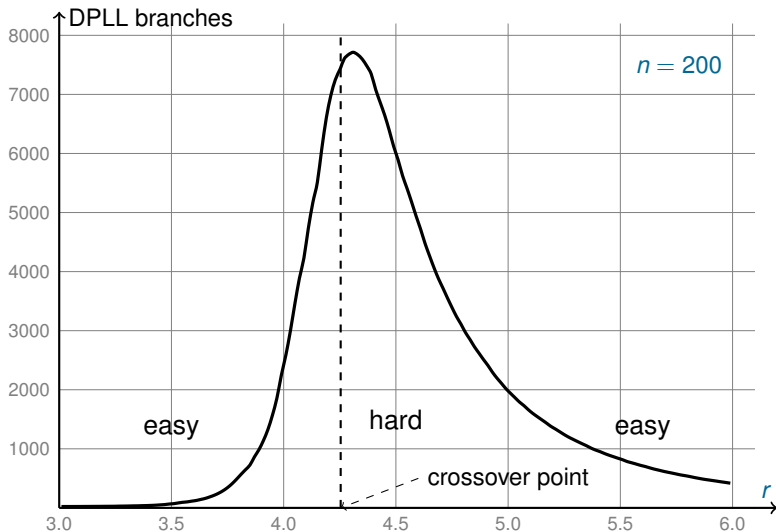




# Easy-Hard-Easy Pattern



# Easy-Hard-Easy Pattern



Experiments show that the crossover point for 3-SAT is 4.25.

# Satisfiability Algorithm that Cannot Establish Unsatisfiability

**procedure** *CHAOS*( $S$ )

**input:** set of clauses  $S$

**output:** interpretation  $I$  such that  $I \models S$  or *don't know*

# Satisfiability Algorithm that Cannot Establish Unsatisfiability

**procedure** *CHAOS*(*S*)

**input:** set of clauses *S*

**output:** interpretation *I* such that  $I \models S$  or *don't know*

**parameters:** positive integer *MAX-TRIES*

**begin**

**repeat** *MAX-TRIES* times

**end**

# Satisfiability Algorithm that Cannot Establish Unsatisfiability

**procedure** *CHAOS*(*S*)

**input:** set of clauses *S*

**output:** interpretation *I* such that  $I \models S$  or *don't know*

**parameters:** positive integer *MAX-TRIES*

**begin**

**repeat** *MAX-TRIES* times

*I* := random interpretation

**if**  $I \models S$  **then return** *I*

**return** *don't know*

**end**

# SAT as a Decision Problem

**Decision problem:** any problem in any infinite domain, that has a **yes-no** answer. Each element of this domain is called an **instance** of this problem.

# SAT as a Decision Problem

**Decision problem:** any problem in any infinite domain, that has a **yes-no** answer. Each element of this domain is called an **instance** of this problem.

Example: solvability of systems of linear inequalities over integers.

- ▶ an **instance** is a system of linear integer inequalities;
- ▶ an answer is **yes** if it the **instance** has a solution.

# SAT as a Decision Problem

**Decision problem:** any problem in any infinite domain, that has a **yes-no** answer. Each element of this domain is called an **instance** of this problem.

Example: solvability of systems of linear inequalities over integers.

- ▶ an **instance** is a system of linear integer inequalities;
- ▶ an answer is **yes** if it the **instance** has a solution.

**SAT is a decision problem:**

- ▶ an **instance** is a finite set of clauses.
- ▶ it has a **yes-no** answer: **yes** (**satisfiable**) or **no** (**unsatisfiable**).



# SAT as a Decision Problem

**Decision problem:** any problem in any infinite domain, that has a **yes-no** answer. Each element of this domain is called an **instance** of this problem.

Example: solvability of systems of linear inequalities over integers.

- ▶ an **instance** is a system of linear integer inequalities;
- ▶ an answer is **yes** if it the **instance** has a solution.

**SAT is a decision problem:**

- ▶ an **instance** is a finite set of clauses.
- ▶ it has a **yes-no** answer: **yes** (satisfiable) or **no** (unsatisfiable).

**Witness for an instance  $\mathcal{I}$ :** any data  $D$  such that, given  $D$ , one can check in polynomial time (in the size of  $D$ ) that  $D$  is a yes-answer of  $\mathcal{I}$ .

# SAT as a Decision Problem

**Decision problem:** any problem in any infinite domain, that has a **yes-no** answer. Each element of this domain is called an **instance** of this problem.

Example: solvability of systems of linear inequalities over integers.

- ▶ an **instance** is a system of linear integer inequalities;
- ▶ an answer is **yes** if it the **instance** has a solution.

**SAT is a decision problem:**

- ▶ an **instance** is a finite set of clauses.
- ▶ it has a **yes-no** answer: **yes** (**satisfiable**) or **no** (**unsatisfiable**).

**Witness for an instance  $\mathcal{I}$ :** any data  $D$  such that, given  $D$ , one can check in polynomial time (in the size of  $D$ ) that  $D$  is a yes-answer of  $\mathcal{I}$ .

Satisfiability has **short witnessess**: interpretations.

# SAT as a Decision Problem

**Decision problem:** any problem in any infinite domain, that has a **yes-no** answer. Each element of this domain is called an **instance** of this problem.

Example: solvability of systems of linear inequalities over integers.

- ▶ an **instance** is a system of linear integer inequalities;
- ▶ an answer is **yes** if it the **instance** has a solution.

**SAT is a decision problem:**

- ▶ an **instance** is a finite set of clauses.
- ▶ it has a **yes-no** answer: **yes** (satisfiable) or **no** (unsatisfiable).

**Witness for an instance  $\mathcal{I}$ :** any data  $D$  such that, given  $D$ , one can check in polynomial time (in the size of  $D$ ) that  $D$  is a yes-answer of  $\mathcal{I}$ .

Satisfiability has **short witnessess**: interpretations.

Unsatisfiability has **no polynomial-size witnessess**, unless  $NP = co - NP$ .

# Randomised Algorithms for SAT

- ▶ Choose a **random interpretation**.

# Randomised Algorithms for SAT

- ▶ Choose a **random interpretation**.
- ▶ If this interpretation is not a model, repeatedly choose a variable and **change its value in the interpretation** (**flip** the variable).

# Randomised Algorithms for SAT

- ▶ Choose a **random interpretation**.
- ▶ If this interpretation is not a model, repeatedly choose a variable and **change its value in the interpretation** (**flip** the variable).

The flipped variables are chosen using heuristics or randomly, or both.

# Randomised Algorithms for SAT

- ▶ Choose a **random interpretation**.
- ▶ If this interpretation is not a model, repeatedly choose a variable and **change its value in the interpretation** (**flip** the variable).

The flipped variables are chosen using heuristics or randomly, or both.

$$\text{flip}(I, p)(q) = \begin{cases} I(q), & \text{if } p \neq q; \\ 1, & \text{if } p = q \text{ and } I(p) = 0; \\ 0, & \text{if } p = q \text{ and } I(p) = 1. \end{cases}$$

# Randomised Algorithms for SAT

- ▶ Choose a **random interpretation**.
- ▶ If this interpretation is not a model, repeatedly choose a variable and **change its value in the interpretation** (**flip** the variable).

The flipped variables are chosen using heuristics or randomly, or both.

$$\text{flip}(I, p)(q) = \begin{cases} I(q), & \text{if } p \neq q; \\ 1, & \text{if } p = q \text{ and } I(p) = 0; \\ 0, & \text{if } p = q \text{ and } I(p) = 1. \end{cases}$$

In other words, the interpretation  $\text{flip}(I, p)$  is obtained from  $I$  by changing its value on  $p$ .



# GSAT

**procedure**  $GSAT(S)$

**input:** set of clauses  $S$

**output:** interpretation  $I$  such that  $I \models S$  or *don't know*

# GSAT

**procedure**  $GSAT(S)$

**input:** set of clauses  $S$

**output:** interpretation  $I$  such that  $I \models S$  or *don't know*

**parameters:** integers  $MAX-TRIES$ ,  $MAX-FLIPS$

# GSAT

procedure *GSAT*(*S*)

input: set of clauses *S*

output: interpretation *I* such that  $I \models S$  or *don't know*

parameters: integers *MAX-TRIES*, *MAX-FLIPS*

begin

repeat *MAX-TRIES* times

*I* := random interpretation

if  $I \models S$  then return *I*

end

# GSAT

**procedure** *GSAT*(*S*)

**input:** set of clauses *S*

**output:** interpretation *I* such that  $I \models S$  or *don't know*

**parameters:** integers *MAX-TRIES*, *MAX-FLIPS*

**begin**

**repeat** *MAX-TRIES* times

*I* := random interpretation

**if**  $I \models S$  **then return** *I*

**repeat** *MAX-FLIPS* times

*p* := a variable such that *flip*(*I*, *p*) satisfies  
the maximal number of clauses in *S*

*I* = *flip*(*I*, *p*)

**if**  $I \models S$  **then return** *I*

**return** *don't know*

**end**

# GSAT example

0		0		1
$p_1$	$\vee$	$\neg p_2$	$\vee$	$p_3$
		$\neg p_2$	$\vee$	$\neg p_3$
$\neg p_1$			$\vee$	$\neg p_3$
$\neg p_1$	$\vee$	$p_2$		
$p_1$	$\vee$	$p_2$		

# GSAT example

	0		0		1
	$p_1$	∨	$\neg p_2$	∨	$p_3$
			$\neg p_2$	∨	$\neg p_3$
	$\neg p_1$			∨	$\neg p_3$
	$\neg p_1$	∨	$p_2$		
	$p_1$	∨	$p_2$		

flip no.	interpretation			satisfied clauses			candidates for flipping	flipped variable
	$p_1$	$p_2$	$p_3$	$p_1$	$p_2$	$p_3$		
1	0	0	1	4				

# GSAT example

0		0		1
$p_1$	$\vee$	$\neg p_2$	$\vee$	$p_3$
		$\neg p_2$	$\vee$	$\neg p_3$
$\neg p_1$			$\vee$	$\neg p_3$
$\neg p_1$	$\vee$	$p_2$		
$p_1$	$\vee$	$p_2$		

flip no.	interpretation			satisfied clauses			candidates for flipping	flipped variable
	$p_1$	$p_2$	$p_3$	$p_1$	$p_2$	$p_3$		
1	0	0	1	4	3	4	4	

# GSAT example

0		1		1
$p_1$	$\vee$	$\neg p_2$	$\vee$	$p_3$
		$\neg p_2$	$\vee$	$\neg p_3$
$\neg p_1$			$\vee$	$\neg p_3$
$\neg p_1$	$\vee$	$p_2$		
$p_1$	$\vee$	$p_2$		

flip no.	interpretation			satisfied clauses				candidates for flipping	flipped variable
	$p_1$	$p_2$	$p_3$	$p_1$	$p_2$	$p_3$			
1	0	0	1	4	3	4	4	$p_2, p_3$	$p_2$
2	0	1	1						



# GSAT example

$$\begin{array}{r}
 \begin{array}{c} 0 \qquad \qquad 1 \qquad \qquad 1 \\ \hline p_1 \vee \neg p_2 \vee p_3 \\ \qquad \qquad \neg p_2 \vee \neg p_3 \\ \neg p_1 \qquad \qquad \vee \neg p_3 \\ \neg p_1 \vee p_2 \\ p_1 \vee p_2 \end{array}
 \end{array}$$

flip no.	interpretation			satisfied clauses			candidates for flipping	flipped variable
	$p_1$	$p_2$	$p_3$	$p_1$	$p_2$	$p_3$		
1	0	0	1	4	3	4	$p_2, p_3$	$p_2$
2	0	1	1	4				

# GSAT example

$$\begin{array}{r}
 \begin{array}{c} 0 \qquad \qquad 1 \qquad \qquad 1 \\ \hline p_1 \vee \neg p_2 \vee p_3 \\ \qquad \qquad \neg p_2 \vee \neg p_3 \\ \neg p_1 \qquad \qquad \vee \neg p_3 \\ \neg p_1 \vee p_2 \\ p_1 \vee p_2 \end{array}
 \end{array}$$

flip no.	interpretation			satisfied clauses				candidates for flipping	flipped variable
	$p_1$	$p_2$	$p_3$		$p_1$	$p_2$	$p_3$		
1	0	0	1	4	3	4	4	$p_2, p_3$	$p_2$
2	0	1	1	4	3	4	4		

# GSAT example

0		1		0
$p_1$	$\vee$	$\neg p_2$	$\vee$	$p_3$
		$\neg p_2$	$\vee$	$\neg p_3$
$\neg p_1$			$\vee$	$\neg p_3$
$\neg p_1$	$\vee$	$p_2$		
$p_1$	$\vee$	$p_2$		

flip no.	interpretation			satisfied clauses				candidates for flipping	flipped variable
	$p_1$	$p_2$	$p_3$	$p_1$	$p_2$	$p_3$			
1	0	0	1	4	3	4	4	$p_2, p_3$	$p_2$
2	0	1	1	4	3	4	4	$p_2, p_3$	$p_3$
3	0	1	0						

# GSAT example

$$\begin{array}{r}
 \begin{array}{c} 0 \\ \hline p_1 \vee \neg p_2 \vee p_3 \\ \neg p_1 \vee \neg p_2 \vee \neg p_3 \\ \neg p_1 \vee p_2 \\ p_1 \vee p_2 \end{array}
 \end{array}$$

flip no.	interpretation			satisfied clauses			candidates for flipping	flipped variable
	$p_1$	$p_2$	$p_3$	$p_1$	$p_2$	$p_3$		
1	0	0	1	4	3	4	$p_2, p_3$	$p_2$
2	0	1	1	4	3	4	$p_2, p_3$	$p_3$
3	0	1	0	4				

# GSAT example

$$\begin{array}{r}
 \begin{array}{c} 0 \\ \hline p_1 \vee \neg p_2 \vee p_3 \\ \neg p_1 \vee \neg p_2 \vee \neg p_3 \\ \neg p_1 \vee p_2 \\ p_1 \vee p_2 \end{array}
 \end{array}$$

flip no.	interpretation			satisfied clauses			candidates for flipping	flipped variable
	$p_1$	$p_2$	$p_3$	$p_1$	$p_2$	$p_3$		
1	0	0	1	4	3	4	$p_2, p_3$	$p_2$
2	0	1	1	4	3	4	$p_2, p_3$	$p_3$
3	0	1	0	4	5	4		

# GSAT example

$$\begin{array}{r}
 \color{red}{1} \qquad \qquad 1 \qquad \qquad 0 \\
 \hline
 p_1 \vee \neg p_2 \vee p_3 \\
 \qquad \qquad \neg p_2 \vee \neg p_3 \\
 \neg p_1 \qquad \qquad \vee \neg p_3 \\
 \neg p_1 \vee p_2 \\
 p_1 \vee p_2
 \end{array}$$

flip no.	interpretation			satisfied clauses				candidates for flipping	flipped variable
	$p_1$	$p_2$	$p_3$		$p_1$	$p_2$	$p_3$		
1	0	0	1	4	3	4	4	$p_2, p_3$	$p_2$
2	0	1	1	4	3	4	4	$p_2, p_3$	$p_3$
3	0	1	0	4	5	4	4	$p_1$	$p_1$
	1	1	0						

# GSAT example

$$\begin{array}{r}
 \begin{array}{c} 1 \\ \hline p_1 \vee \neg p_2 \vee p_3 \\ \neg p_1 \vee \neg p_2 \vee \neg p_3 \\ \neg p_1 \vee p_2 \\ p_1 \vee p_2 \end{array}
 \end{array}$$

flip no.	interpretation			satisfied clauses			candidates for flipping	flipped variable	
	$p_1$	$p_2$	$p_3$	$p_1$	$p_2$	$p_3$			
1	0	0	1	4	3	4	4	$p_2, p_3$	$p_2$
2	0	1	1	4	3	4	4	$p_2, p_3$	$p_3$
3	0	1	0	4	5	4	4	$p_1$	$p_1$
	1	1	0	5					

# WSAT

**procedure**  $WSAT(S)$

**input:** set of clauses  $S$

**output:** interpretation  $I$  such that  $I \models S$  or *don't know*

**parameters:** integers  $MAX-TRIES$ ,  $MAX-FLIPS$



# WSAT

procedure *WSAT(S)*

input: set of clauses *S*

output: interpretation *I* such that  $I \models S$  or *don't know*

parameters: integers *MAX-TRIES*, *MAX-FLIPS*

begin

repeat *MAX-TRIES* times

*I* := random interpretation

if  $I \models S$  then return *I*

end

# WSAT

**procedure** *WSAT*(*S*)

**input:** set of clauses *S*

**output:** interpretation *I* such that  $I \models S$  or *don't know*

**parameters:** integers *MAX-TRIES*, *MAX-FLIPS*

**begin**

**repeat** *MAX-TRIES* times

*I* := random interpretation

**if**  $I \models S$  **then return** *I*

**repeat** *MAX-FLIPS* times

randomly select a clause  $C \in S$  such that  $I \not\models C$

randomly select a variable *p* in *C*

*I* = *flip*(*I*, *p*)

**if**  $I \models S$  **then return** *I*

**return** *don't know*

**end**

# WSAT example

0		0		1
$p_1$	$\vee$	$\neg p_2$	$\vee$	$p_3$
		$\neg p_2$	$\vee$	$\neg p_3$
$\neg p_1$			$\vee$	$\neg p_3$
$\neg p_1$	$\vee$	$p_2$		
$p_1$	$\vee$	$p_2$		

# WSAT example

0		0		1
$p_1$	∨	$\neg p_2$	∨	$p_3$
		$\neg p_2$	∨	$\neg p_3$
$\neg p_1$			∨	$\neg p_3$
$\neg p_1$	∨	$p_2$		
$p_1$	∨	$p_2$		

flip no.	interpretation			unsatisfied clauses	candidates for flipping	flipped variable
	$p_1$	$p_2$	$p_3$			
1	0	0	1			

# WSAT example

$$\begin{array}{r}
 \begin{array}{c} 0 \\ \hline p_1 \vee \neg p_2 \\ \neg p_1 \vee p_2 \\ p_1 \vee p_2 \end{array}
 \quad
 \begin{array}{c} 0 \\ \hline \neg p_2 \vee \neg p_3 \\ \neg p_2 \vee \neg p_3 \\ \vee \\ p_2 \end{array}
 \quad
 \begin{array}{c} 1 \\ \hline p_3 \\ \neg p_3 \\ \vee \\ \neg p_3 \end{array}
 \end{array}$$

flip no.	interpretation			unsatisfied clauses	candidates for flipping	flipped variable
	$p_1$	$p_2$	$p_3$			
1	0	0	1	$p_1 \vee p_2$	$p_1, p_2$	

# WSAT example

$$\begin{array}{rcc}
 1 & & 0 & & 1 \\
 \hline
 p_1 & \vee & \neg p_2 & \vee & p_3 \\
 & & \neg p_2 & \vee & \neg p_3 \\
 & & & \vee & \neg p_3 \\
 \neg p_1 & & & & \\
 \neg p_1 & \vee & p_2 & & \\
 p_1 & \vee & p_2 & & 
 \end{array}$$

flip no.	interpretation			unsatisfied clauses	candidates for flipping	flipped variable
	$p_1$	$p_2$	$p_3$			
1	0	0	1	$p_1 \vee p_2$	$p_1, p_2$	$p_1$
2	1	0	1			

# WSAT example

1		0		1
$p_1$	$\vee$	$\neg p_2$	$\vee$	$p_3$
		$\neg p_2$	$\vee$	$\neg p_3$
$\neg p_1$			$\vee$	$\neg p_3$
$\neg p_1$	$\vee$	$p_2$		
$p_1$	$\vee$	$p_2$		

flip no.	interpretation			unsatisfied clauses	candidates for flipping	flipped variable
	$p_1$	$p_2$	$p_3$			
1	0	0	1	$p_1 \vee p_2$	$p_1, p_2$	$p_1$
2	1	0	1	$\neg p_1 \vee \neg p_3$ $\neg p_1 \vee p_2$	$p_1, p_2, p_3$	

# WSAT example

$$\begin{array}{r}
 1 \qquad \qquad 1 \qquad \qquad 1 \\
 \hline
 p_1 \vee \neg p_2 \vee p_3 \\
 \qquad \qquad \neg p_2 \vee \neg p_3 \\
 \neg p_1 \qquad \qquad \vee \neg p_3 \\
 \neg p_1 \vee p_2 \\
 p_1 \vee p_2
 \end{array}$$

flip no.	interpretation			unsatisfied clauses	candidates for flipping	flipped variable
	$p_1$	$p_2$	$p_3$			
1	0	0	1	$p_1 \vee p_2$	$p_1, p_2$	$p_1$
2	1	0	1	$\neg p_1 \vee \neg p_3$ $\neg p_1 \vee p_2$	$p_1, p_2, p_3$	$p_2$
3	1	1	1			



# WSAT example

$$\begin{array}{r}
 1 \qquad \qquad 1 \qquad \qquad 1 \\
 \hline
 p_1 \vee \neg p_2 \vee p_3 \\
 \qquad \qquad \neg p_2 \vee \neg p_3 \\
 \neg p_1 \qquad \qquad \vee \neg p_3 \\
 \neg p_1 \vee p_2 \\
 p_1 \vee p_2
 \end{array}$$

flip no.	interpretation			unsatisfied clauses	candidates for flipping	flipped variable
	$p_1$	$p_2$	$p_3$			
1	0	0	1	$p_1 \vee p_2$	$p_1, p_2$	$p_1$
2	1	0	1	$\neg p_1 \vee \neg p_3$ $\neg p_1 \vee p_2$	$p_1, p_2, p_3$	$p_2$
3	1	1	1	$\neg p_2 \vee \neg p_3$ $\neg p_1 \vee \neg p_3$	$p_1, p_2, p_3$	

# WSAT example

$$\begin{array}{r}
 1 \qquad \qquad 1 \qquad \qquad 0 \\
 \hline
 p_1 \vee \neg p_2 \vee p_3 \\
 \qquad \qquad \neg p_2 \vee \neg p_3 \\
 \neg p_1 \qquad \qquad \vee \neg p_3 \\
 \neg p_1 \vee p_2 \\
 p_1 \vee p_2
 \end{array}$$

flip no.	interpretation			unsatisfied clauses	candidates for flipping	flipped variable
	$p_1$	$p_2$	$p_3$			
1	0	0	1	$p_1 \vee p_2$	$p_1, p_2$	$p_1$
2	1	0	1	$\neg p_1 \vee \neg p_3$ $\neg p_1 \vee p_2$	$p_1, p_2, p_3$	$p_2$
3	1	1	1	$\neg p_2 \vee \neg p_3$ $\neg p_1 \vee \neg p_3$	$p_1, p_2, p_3$	$p_3$
	1	1	0			

# WSAT example

$$\begin{array}{r}
 \begin{array}{c} 1 \\ \hline p_1 \end{array} \vee \begin{array}{c} 1 \\ \hline \neg p_2 \end{array} \vee \begin{array}{c} 0 \\ \hline p_3 \\ \neg p_3 \end{array} \\
 \neg p_1 \vee \begin{array}{c} 1 \\ \hline p_2 \\ p_2 \end{array} \\
 \neg p_1 \vee \begin{array}{c} 1 \\ \hline p_2 \end{array}
 \end{array}$$

flip no.	interpretation			unsatisfied clauses	candidates for flipping	flipped variable
	$p_1$	$p_2$	$p_3$			
1	0	0	1	$p_1 \vee p_2$	$p_1, p_2$	$p_1$
2	1	0	1	$\neg p_1 \vee \neg p_3$ $\neg p_1 \vee p_2$	$p_1, p_2, p_3$	$p_2$
3	1	1	1	$\neg p_2 \vee \neg p_3$ $\neg p_1 \vee \neg p_3$	$p_1, p_2, p_3$	$p_3$
	1	1	0			

# Outline

## Satisfiability and Randomisation

Randomly Generated Clause Sets

Sharp Phase Transition

Randomised Algorithms for Satisfiability-Checking

## Horn clauses

# Horn clauses

A clause is called **Horn** if it contains at most one positive literal.

Examples:

$$\begin{aligned} & p_1 \\ \neg p_1 \vee p_2 \\ \neg p_1 \vee \neg p_2 \vee p_3 \\ \neg p_3 \vee \neg p_4 \end{aligned}$$

# Horn clauses

A clause is called **Horn** if it contains at most one positive literal.

Examples:

$$\begin{aligned} & p_1 \\ & \neg p_1 \vee p_2 \\ & \neg p_1 \vee \neg p_2 \vee p_3 \\ & \neg p_3 \vee \neg p_4 \end{aligned}$$

The following clauses are non-Horn:

$$\begin{aligned} & p_1 \vee p_2 \\ & p_1 \vee \neg p_2 \vee p_3 \end{aligned}$$

# Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\begin{aligned} & p_1 \\ & \neg p_1 \vee p_2 \\ & \neg p_1 \vee \neg p_2 \vee p_3 \\ & \neg p_3 \vee \neg p_4 \end{aligned}$$

# Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\begin{aligned} & p_1 \\ & \neg p_1 \vee p_2 \\ & \neg p_1 \vee \neg p_2 \vee p_3 \\ & \neg p_3 \vee \neg p_4 \end{aligned}$$



# Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\begin{array}{c} p_2 \\ \neg p_2 \vee p_3 \\ \neg p_3 \vee \neg p_4 \end{array}$$

# Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\neg p_3 \vee \overset{p_3}{\neg p_4}$$

# Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\neg p_4$$

# Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\begin{aligned} & p_1 \\ & \neg p_1 \vee p_2 \\ & \neg p_1 \vee \neg p_2 \vee p_3 \\ & \neg p_3 \vee \neg p_4 \end{aligned}$$

Model:  $\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 1, p_4 \mapsto 0\}$

# Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\begin{aligned} & p_1 \\ & \neg p_1 \vee p_2 \\ & \neg p_1 \vee \neg p_2 \vee p_3 \\ & \neg p_3 \vee \neg p_4 \end{aligned}$$

Model:  $\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 1, p_4 \mapsto 0\}$

Note that deleting a literal from a Horn clause gives a Horn clause.

# Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\begin{aligned} & p_1 \\ & \neg p_1 \vee p_2 \\ & \neg p_1 \vee \neg p_2 \vee p_3 \\ & \neg p_3 \vee \neg p_4 \end{aligned}$$

Model:  $\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 1, p_4 \mapsto 0\}$

Note that deleting a literal from a Horn clause gives a Horn clause.

Therefore, unit propagation applied to a set  $C$  of Horn clauses gives a set  $C'$  of Horn clauses.

# Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\begin{aligned} & p_1 \\ & \neg p_1 \vee p_2 \\ & \neg p_1 \vee \neg p_2 \vee p_3 \\ & \neg p_3 \vee \neg p_4 \end{aligned}$$

Model:  $\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 1, p_4 \mapsto 0\}$

Note that deleting a literal from a Horn clause gives a Horn clause.

Therefore, unit propagation applied to a set  $C$  of Horn clauses gives a set  $C'$  of Horn clauses.

Two cases:

1.  $C'$  contains  $\square$ . Then,  $C'$  (and hence  $C$ ) is **unsatisfiable**.

# Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\begin{aligned} & p_1 \\ & \neg p_1 \vee p_2 \\ & \neg p_1 \vee \neg p_2 \vee p_3 \\ & \neg p_3 \vee \neg p_4 \end{aligned}$$

Model:  $\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 1, p_4 \mapsto 0\}$

Note that deleting a literal from a Horn clause gives a Horn clause.

Therefore, unit propagation applied to a set  $C$  of Horn clauses gives a set  $C'$  of Horn clauses.

Two cases:

1.  $C'$  contains  $\square$ . Then,  $C'$  (and hence  $C$ ) is **unsatisfiable**.
2.  $C'$  does not contain  $\square$ .



# Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\begin{aligned} & p_1 \\ & \neg p_1 \vee p_2 \\ & \neg p_1 \vee \neg p_2 \vee p_3 \\ & \neg p_3 \vee \neg p_4 \end{aligned}$$

Model:  $\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 1, p_4 \mapsto 0\}$

Note that deleting a literal from a Horn clause gives a Horn clause.

Therefore, unit propagation applied to a set  $C$  of Horn clauses gives a set  $C'$  of Horn clauses.

Two cases:

1.  $C'$  contains  $\square$ . Then,  $C'$  (and hence  $C$ ) is **unsatisfiable**.
2.  $C'$  does not contain  $\square$ . Then:
  - ▶ Each clause in  $C'$  has at least two literals.

# Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\begin{aligned} & p_1 \\ & \neg p_1 \vee p_2 \\ & \neg p_1 \vee \neg p_2 \vee p_3 \\ & \neg p_3 \vee \neg p_4 \end{aligned}$$

Model:  $\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 1, p_4 \mapsto 0\}$

Note that deleting a literal from a Horn clause gives a Horn clause.

Therefore, unit propagation applied to a set  $C$  of Horn clauses gives a set  $C'$  of Horn clauses.

Two cases:

1.  $C'$  contains  $\square$ . Then,  $C'$  (and hence  $C$ ) is **unsatisfiable**.
2.  $C'$  does not contain  $\square$ . Then:
  - ▶ Each clause in  $C'$  has at least two literals.
  - ▶ Hence each clause in  $C'$  contains at least one negative literal;

# Satisfiability of Horn clauses can be decided by unit propagation

Example:

$$\begin{aligned} & p_1 \\ & \neg p_1 \vee p_2 \\ & \neg p_1 \vee \neg p_2 \vee p_3 \\ & \neg p_3 \vee \neg p_4 \end{aligned}$$

Model:  $\{p_1 \mapsto 1, p_2 \mapsto 1, p_3 \mapsto 1, p_4 \mapsto 0\}$

Note that deleting a literal from a Horn clause gives a Horn clause.

Therefore, unit propagation applied to a set  $C$  of Horn clauses gives a set  $C'$  of Horn clauses.

Two cases:

1.  $C'$  contains  $\square$ . Then,  $C'$  (and hence  $C$ ) is **unsatisfiable**.
2.  $C'$  does not contain  $\square$ . Then:
  - ▶ Each clause in  $C'$  has at least two literals.
  - ▶ Hence each clause in  $C'$  contains at least one negative literal;
  - ▶ Hence setting all variables in  $C'$  to 0 satisfies  $C'$ .