

# Satisfiability

Christoph Schwering

COMP4418, Week 6 & 7

# The *Reasoning* in KRR

We've touched *reasoning* a few times so far:

- On a theoretical level:

- ▶ Resolution

- Prove that KB entails  $\psi$  by deriving contradiction from  $KB \wedge \neg\psi$ .
    - Downside: Difficult to guide search towards goal.

- ▶ Prolog

- Uses restricted version of resolution.
    - Downside: More programming than reasoning.

# The Reasoning in KRR

We've touched reasoning a few times so far:

- On a theoretical level:

- ▶ Resolution

- Prove that KB entails  $\psi$  by deriving contradiction from  $KB \wedge \neg\psi$ .
    - Downside: Difficult to guide search towards goal.

- ▶ Prolog

- Uses restricted version of resolution.
    - Downside: More programming than reasoning.

- From a user perspective:

- ▶ Answer Set Programming

- ▶ Satisfiability (in Assignment 1, Question 3)

# The Reasoning in KRR

We've touched reasoning a few times so far:

- On a theoretical level:

- ▶ Resolution

- Prove that KB entails  $\psi$  by deriving contradiction from  $KB \wedge \neg\psi$ .
    - Downside: Difficult to guide search towards goal.

- ▶ Prolog

- Uses restricted version of resolution.
    - Downside: More programming than reasoning.

- From a user perspective:

- ▶ Answer Set Programming

- ▶ Satisfiability (in Assignment 1, Question 3)

Today: propositional reasoning

- What algorithms?

- What do real-world implementations look like?

- Which problems can be solved using it (in theory & practice)?

# Satisfiability

## Definition: SAT

Input: Propositional formula in CNF.

Problem: Is this formula satisfiable?

- Many problems can be reduced to SAT.
- In theory, SAT is very difficult.
- In practice, SAT is often feasible.
- SAT is extremely important for theory *and* practice.

This lecture: only propositional logic.

# This Lecture

The goal of this lecture is twofold:

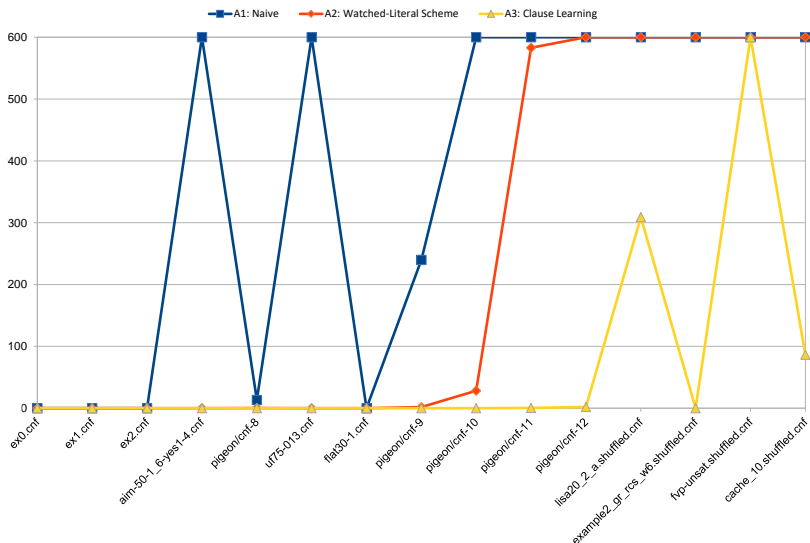
- Practical aspects:

- ▶ What are data structures / algorithms for SAT solving?
- ▶ What does an implementation of a SAT solver look like?

- Theoretical aspects:

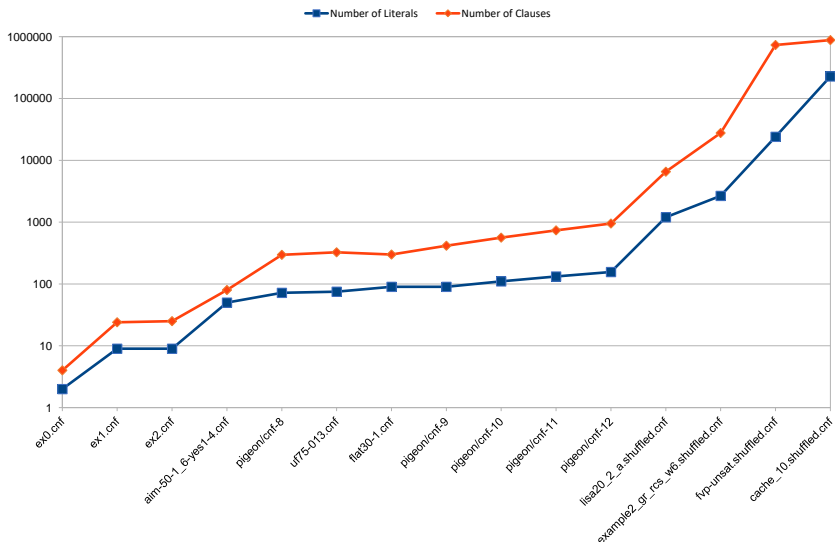
- ▶ Why is SAT hard?
- ▶ Why is it so important?

# Impact of Advanced Algorithms



Running times (in seconds) of algorithms for different instances.

# Impact of Advanced Algorithms



Size of the instances.



# Outline

Satisfiability and Complexity Theory

Solver Ingredient 1: Watched-Literal Scheme

Solver Ingredient 2: Conflict-Driven Clause Learning

Other Solver Ingredients

Conclusion

Beyond SAT

# Outline

Satisfiability and Complexity Theory

Solver Ingredient 1: Watched-Literal Scheme

Solver Ingredient 2: Conflict-Driven Clause Learning

Other Solver Ingredients

Conclusion

Beyond SAT

# Propositional Logic

## Syntax:

- Atomic propositions a.k.a. **variables**.
- Negation:  $\neg\phi$
- Disjunction:  $(\phi_1 \vee \phi_2)$
- Conjunction:  $(\phi_1 \wedge \phi_2)$
- TRUE, FALSE,  $\rightarrow$ ,  $\leftrightarrow$  can be expressed with  $\neg$ ,  $\vee$ ,  $\wedge$ .

## Semantics:

- An interpretation  $I$  maps each variable to true or false.
- It satisfies  $\neg\phi$  iff it falsifies<sup>1</sup>  $\phi$ .
- It satisfies  $(\phi_1 \vee \phi_2)$  iff it satisfies  $\phi_1$  or  $\phi_2$ .
- It satisfies  $(\phi_1 \wedge \phi_2)$  iff it satisfies  $\phi_1$  and  $\phi_2$ .

---

<sup>1</sup>falsify = do not satisfy

# Satisfiability and Validity

- $\phi$  is satisfiable iff some interpretation satisfies  $\phi$
- $\phi$  is valid iff all interpretations satisfy  $\phi$

# Satisfiability and Validity

- $\phi$  is satisfiable iff some interpretation satisfies  $\phi$   
iff not: not: some interpretation satisfies  $\phi$
- $\phi$  is valid iff all interpretations satisfy  $\phi$

# Satisfiability and Validity

- $\phi$  is satisfiable iff some interpretation satisfies  $\phi$   
iff not: not: some interpretation satisfies  $\phi$   
iff not: all interpretations falsify  $\phi$
  
- $\phi$  is valid iff all interpretations satisfy  $\phi$

# Satisfiability and Validity

- $\phi$  is satisfiable iff some interpretation satisfies  $\phi$   
iff not: not: some interpretation satisfies  $\phi$   
iff not: all interpretations falsify  $\phi$   
iff not: all interpretations satisfy  $\neg\phi$
- $\phi$  is valid iff all interpretations satisfy  $\phi$

# Satisfiability and Validity

- $\phi$  is satisfiable iff some interpretation satisfies  $\phi$   
iff not: not: some interpretation satisfies  $\phi$   
iff not: all interpretations falsify  $\phi$   
iff not: all interpretations satisfy  $\neg\phi$   
iff  $\neg\phi$  is not valid
- $\phi$  is valid iff all interpretations satisfy  $\phi$



# Satisfiability and Validity

- $\phi$  is satisfiable iff some interpretation satisfies  $\phi$   
iff not: not: some interpretation satisfies  $\phi$   
iff not: all interpretations falsify  $\phi$   
iff not: all interpretations satisfy  $\neg\phi$   
iff  $\neg\phi$  is not valid
- $\phi$  is valid iff all interpretations satisfy  $\phi$   
iff not: not: all interpretation satisfy  $\phi$

# Satisfiability and Validity

- $\phi$  is satisfiable iff some interpretation satisfies  $\phi$   
iff not: not: some interpretation satisfies  $\phi$   
iff not: all interpretations falsify  $\phi$   
iff not: all interpretations satisfy  $\neg\phi$   
iff  $\neg\phi$  is not valid
- $\phi$  is valid iff all interpretations satisfy  $\phi$   
iff not: not: all interpretation satisfy  $\phi$   
iff not: some interpretation falsifies  $\phi$

# Satisfiability and Validity

- $\phi$  is satisfiable iff some interpretation satisfies  $\phi$   
iff not: not: some interpretation satisfies  $\phi$   
iff not: all interpretations falsify  $\phi$   
iff not: all interpretations satisfy  $\neg\phi$   
iff  $\neg\phi$  is not valid
- $\phi$  is valid iff all interpretations satisfy  $\phi$   
iff not: not: all interpretation satisfy  $\phi$   
iff not: some interpretation falsifies  $\phi$   
iff not: some interpretation satisfies  $\neg\phi$

# Satisfiability and Validity

- $\phi$  is satisfiable iff some interpretation satisfies  $\phi$   
iff not: not: some interpretation satisfies  $\phi$   
iff not: all interpretations falsify  $\phi$   
iff not: all interpretations satisfy  $\neg\phi$   
iff  $\neg\phi$  is not valid
- $\phi$  is valid iff all interpretations satisfy  $\phi$   
iff not: not: all interpretation satisfy  $\phi$   
iff not: some interpretation falsifies  $\phi$   
iff not: some interpretation satisfies  $\neg\phi$   
iff  $\neg\phi$  is not satisfiable

# Disjunctive Normal Form

## Definition: DNF

A formula  $\psi$  is in Disjunctive Normal Form iff

$\psi$  is of the form  $(d_1 \vee \dots \vee d_k)$ , where each  $d_i$  is of the form  $(x_{i1} \wedge \dots \wedge x_{il_i})$ , where each  $x_{ij}$  is a literal.

Ex.:  $(p \wedge q) \vee (\neg p \wedge \neg q) \vee (p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge r)$

Every formula can be converted into an equivalent formula in DNF.  
(Convert into NNF, then use transitivity and distributivity of  $\wedge$  and  $\vee$ .)

Note: Satisfiability for DNF is very easy (solvable in linear time).

# Conjunctive Normal Form

## Definition: CNF

A formula  $\phi$  is in Conjunctive Normal Form iff  
 $\phi$  is of the form  $(c_1 \wedge \dots \wedge c_k)$ , where each  
 $c_i$  is of the form  $(x_{i1} \vee \dots \vee x_{il_i})$ , where each  
 $x_{ij}$  is a literal.

Ex.:  $(\neg p \vee \neg q) \wedge (p \vee q) \wedge (\neg p \vee q \vee \neg r) \wedge (p \vee \neg q \vee \neg r)$

Every formula can be converted into an equivalent formula in CNF.  
(Convert into NNF, then use transitivity and distributivity of  $\wedge$  and  $\vee$ .)

Note: Validity for CNF is very easy (solvable in linear time).

# SAT and CNF

## Definition: SAT

Input:  $\phi$  in CNF.

Problem: Is this formula satisfiable?

Why not DNF?

- Satisfiability for DNF is very easy, but ...
- DNF of formula may grow exponentially!

Why CNF?

- Structure of CNF is much simpler than of arbitrary formulas.
- Every formula  $\psi$  can be transformed to a formula  $\phi$  such that:
  - ▶  $\phi$  is satisfiable iff  $\psi$  is satisfiable.
  - ▶ The size of  $\phi$  is linear in the size of  $\psi$ .

## Tseitin transformation: Example

Input:  $\psi = (\neg((p \vee q) \wedge r) \vee \neg s)$



## Tseitin transformation: Example

Input:  $\psi = (\neg((p \vee q) \wedge r) \vee \neg s)$

$(\neg((p \vee q) \wedge r) \vee \neg s)$

$p$

$q$

$r$

$s$

$\neg s$

$(p \vee q)$

$((p \vee q) \wedge r)$

$\neg((p \vee q) \wedge r)$

$(\neg((p \vee q) \wedge r) \vee \neg s)$

## Tseitin transformation: Example

Input:  $\psi = (\neg((p \vee q) \wedge r) \vee \neg s)$

$$(\neg((p \vee q) \wedge r) \vee \neg s)$$

$p$

$$x_1 \leftrightarrow p$$

$q$

$$x_2 \leftrightarrow q$$

$r$

$$x_3 \leftrightarrow r$$

$s$

$$x_4 \leftrightarrow s$$

$\neg s$

$$x_5 \leftrightarrow \neg s$$

$$(p \vee q)$$

$$x_6 \leftrightarrow (p \vee q)$$

$$((p \vee q) \wedge r)$$

$$x_7 \leftrightarrow ((p \vee q) \wedge r)$$

$$\neg((p \vee q) \wedge r)$$

$$x_8 \leftrightarrow \neg((p \vee q) \wedge r)$$

$$(\neg((p \vee q) \wedge r) \vee \neg s)$$

$$x_9 \leftrightarrow (\neg((p \vee q) \wedge r) \vee \neg s)$$

## Tseitin transformation: Example

Input:  $\psi = (\neg((p \vee q) \wedge r) \vee \neg s)$

$$(\neg((p \vee q) \wedge r) \vee \neg s)$$

$p$

$$x_1 \leftrightarrow p$$

$q$

$$x_2 \leftrightarrow q$$

$r$

$$x_3 \leftrightarrow r$$

$s$

$$x_4 \leftrightarrow s$$

$\neg s$

$$x_5 \leftrightarrow \neg x_4$$

$$(p \vee q)$$

$$x_6 \leftrightarrow (p \vee q)$$

$$((p \vee q) \wedge r)$$

$$x_7 \leftrightarrow ((p \vee q) \wedge r)$$

$$\neg((p \vee q) \wedge r)$$

$$x_8 \leftrightarrow \neg((p \vee q) \wedge r)$$

$$(\neg((p \vee q) \wedge r) \vee \neg s)$$

$$x_9 \leftrightarrow (\neg((p \vee q) \wedge r) \vee \neg s)$$

## Tseitin transformation: Example

Input:  $\psi = (\neg((p \vee q) \wedge r) \vee \neg s)$

$$(\neg((p \vee q) \wedge r) \vee \neg s)$$

$p$

$$x_1 \leftrightarrow p$$

$q$

$$x_2 \leftrightarrow q$$

$r$

$$x_3 \leftrightarrow r$$

$s$

$$x_4 \leftrightarrow s$$

$\neg s$

$$x_5 \leftrightarrow \neg s$$

$$(p \vee q)$$

$$x_6 \leftrightarrow (x_1 \vee x_2)$$

$$((p \vee q) \wedge r)$$

$$x_7 \leftrightarrow ((p \vee q) \wedge r)$$

$$\neg((p \vee q) \wedge r)$$

$$x_8 \leftrightarrow \neg((p \vee q) \wedge r)$$

$$(\neg((p \vee q) \wedge r) \vee \neg s)$$

$$x_9 \leftrightarrow (\neg((p \vee q) \wedge r) \vee \neg s)$$

## Tseitin transformation: Example

Input:  $\psi = (\neg((p \vee q) \wedge r) \vee \neg s)$

$$(\neg((p \vee q) \wedge r) \vee \neg s)$$

$p$

$$x_1 \leftrightarrow p$$

$q$

$$x_2 \leftrightarrow q$$

$r$

$$x_3 \leftrightarrow r$$

$s$

$$x_4 \leftrightarrow s$$

$\neg s$

$$x_5 \leftrightarrow \neg s$$

$$(p \vee q)$$

$$x_6 \leftrightarrow (x_1 \vee x_2)$$

$$((p \vee q) \wedge r)$$

$$x_7 \leftrightarrow (x_6 \wedge x_3)$$

$$\neg((p \vee q) \wedge r)$$

$$x_8 \leftrightarrow \neg((p \vee q) \wedge r)$$

$$(\neg((p \vee q) \wedge r) \vee \neg s)$$

$$x_9 \leftrightarrow (\neg((p \vee q) \wedge r) \vee \neg s)$$

## Tseitin transformation: Example

Input:  $\psi = (\neg((p \vee q) \wedge r) \vee \neg s)$

$$(\neg((p \vee q) \wedge r) \vee \neg s)$$

$p$

$$x_1 \leftrightarrow p$$

$q$

$$x_2 \leftrightarrow q$$

$r$

$$x_3 \leftrightarrow r$$

$s$

$$x_4 \leftrightarrow s$$

$\neg s$

$$x_5 \leftrightarrow \neg s$$

$$(p \vee q)$$

$$x_6 \leftrightarrow (x_1 \vee x_2)$$

$$((p \vee q) \wedge r)$$

$$x_7 \leftrightarrow (x_6 \wedge x_3)$$

$$\neg((p \vee q) \wedge r)$$

$$x_8 \leftrightarrow \neg x_7$$

$$(\neg((p \vee q) \wedge r) \vee \neg s)$$

$$x_9 \leftrightarrow (\neg((p \vee q) \wedge r) \vee \neg s)$$

## Tseitin transformation: Example

Input:  $\psi = (\neg((p \vee q) \wedge r) \vee \neg s)$

$$(\neg((p \vee q) \wedge r) \vee \neg s)$$

$p$

$$x_1 \leftrightarrow p$$

$q$

$$x_2 \leftrightarrow q$$

$r$

$$x_3 \leftrightarrow r$$

$s$

$$x_4 \leftrightarrow s$$

$\neg s$

$$x_5 \leftrightarrow \neg s$$

$$(p \vee q)$$

$$x_6 \leftrightarrow (x_1 \vee x_2)$$

$$((p \vee q) \wedge r)$$

$$x_7 \leftrightarrow (x_6 \wedge x_3)$$

$$\neg((p \vee q) \wedge r)$$

$$x_8 \leftrightarrow \neg x_7$$

$$(\neg((p \vee q) \wedge r) \vee \neg s)$$

$$x_9 \leftrightarrow (x_8 \vee x_5)$$

## Tseitin transformation: Example

Input:  $\psi = (\neg((p \vee q) \wedge r) \vee \neg s)$

$(\neg((p \vee q) \wedge r) \vee \neg s)$

$$\begin{aligned}\phi = & \text{CNF}[x_1 \leftrightarrow p] \\ & \wedge \text{CNF}[x_2 \leftrightarrow q] \\ & \wedge \text{CNF}[x_3 \leftrightarrow r] \\ & \wedge \text{CNF}[x_4 \leftrightarrow s] \\ & \wedge \text{CNF}[x_5 \leftrightarrow \neg s] \\ & \wedge \text{CNF}[x_6 \leftrightarrow (x_1 \vee x_2)] \\ & \wedge \text{CNF}[x_7 \leftrightarrow (x_6 \wedge x_3)] \\ & \wedge \text{CNF}[x_8 \leftrightarrow \neg x_7] \\ & \wedge \text{CNF}[x_9 \leftrightarrow (x_8 \vee x_5)] \\ & \wedge x_9\end{aligned}$$



## Tseitin transformation: Example

Input:  $\psi = (\neg((p \vee q) \wedge r) \vee \neg s)$

$$(\neg((p \vee q) \wedge r) \vee \neg s)$$

$$\begin{aligned}\phi = & (\neg x_1 \vee p) \wedge (x_1 \vee \neg p) \\ & \wedge (\neg x_2 \vee q) \wedge (x_2 \vee \neg q) \\ & \wedge (\neg x_3 \vee r) \wedge (x_3 \vee \neg r) \\ & \wedge (\neg x_4 \vee s) \wedge (x_4 \vee \neg s) \\ & \wedge (\neg x_5 \vee \neg x_4) \wedge (x_5 \vee \neg x_4) \\ & \wedge (\neg x_6 \vee x_1 \vee x_2) \wedge (x_6 \vee \neg x_1) \wedge (x_6 \vee \neg x_2) \\ & \wedge (\neg x_7 \vee x_6) \wedge (\neg x_7 \vee x_3) \wedge (x_7 \vee \neg x_6 \vee \neg x_3) \\ & \wedge (\neg x_8 \vee \neg x_7) \wedge (x_8 \vee x_7) \\ & \wedge (\neg x_9 \vee x_8 \vee x_5) \wedge (x_9 \vee \neg x_8) \wedge (x_9 \vee \neg x_5) \\ & \wedge x_9\end{aligned}$$

# Tseitin Transformation

Input: formula  $\psi$ . Output: formula  $\phi$  in CNF.

- Let  $\rho_1, \dots, \rho_k$  be all sub-formulas of  $\psi$ , where  $\rho_k = \psi$ .
- Let  $x_1, \dots, x_k$  be fresh variables.
- Let  $\phi = \text{CNF}[x_1 \leftrightarrow f(\rho_1)] \wedge \dots \wedge \text{CNF}[x_k \leftrightarrow f(\rho_k)] \wedge x_k$  where
  - ▶  $f(\rho_i \wedge \rho_j) = (x_i \wedge x_j)$
  - ▶  $f(\rho_i \vee \rho_j) = (x_i \vee x_j)$
  - ▶  $f(\neg \rho_i) = \neg x_i$
  - ▶  $f(\rho_i) = x_i$  if  $\rho_i$  is a variable.

# Tseitin Transformation

Input: formula  $\psi$ . Output: formula  $\phi$  in CNF.

- Let  $\rho_1, \dots, \rho_k$  be all sub-formulas of  $\psi$ , where  $\rho_k = \psi$ .
- Let  $x_1, \dots, x_k$  be fresh variables.
- Let  $\phi = \text{CNF}[x_1 \leftrightarrow f(\rho_1)] \wedge \dots \wedge \text{CNF}[x_k \leftrightarrow f(\rho_k)] \wedge x_k$  where
  - ▶  $f(\rho_i \wedge \rho_j) = (x_i \wedge x_j)$
  - ▶  $f(\rho_i \vee \rho_j) = (x_i \vee x_j)$
  - ▶  $f(\neg \rho_i) = \neg x_i$
  - ▶  $f(\rho_i) = x_i$  if  $\rho_i$  is a variable.

## Theorem: Tseitin transformation

$\phi$  is satisfiable iff  $\psi$  is satisfiable.

$\phi$  is in 3-CNF and its size is linear in the size of  $\psi$ .

Why? The number of subformulas of  $\psi$  is at most twice the length  $\psi$ .

The size of  $f(\rho_i)$  is constant. The size of  $\text{CNF}[x_i \leftrightarrow f(\rho_i)]$  is constant.

# Syntax and Semantics Revisited

Syntax:

- A CNF formula  $\phi$  is of the form  $(c_1 \wedge \dots \wedge c_k)$ , where each  $c_i$  is of the form  $(x_{i1} \vee \dots \vee x_{il_i})$  where each  $x_{ij}$  is a variable or a negated variable.
- We identify  $c_i$  with the set  $\{x_{i1}, \dots, x_{il_i}\}$ .
- We identify  $\phi$  with the set  $\{c_1, \dots, c_k\}$ .
- We write  $\bar{x}$  to flip the negation of a literal  $x$ :  $\overline{\overline{p}} = p$   $\overline{p} = \neg p$

# Syntax and Semantics Revisited

## Syntax:

- A CNF formula  $\phi$  is of the form  $(c_1 \wedge \dots \wedge c_k)$ , where each  $c_i$  is of the form  $(x_{i1} \vee \dots \vee x_{il_i})$  where each  $x_{ij}$  is a variable or a negated variable.
- We identify  $c_i$  with the set  $\{x_{i1}, \dots, x_{il_i}\}$ .
- We identify  $\phi$  with the set  $\{c_1, \dots, c_k\}$ .
- We write  $\bar{x}$  to flip the negation of a literal  $x$ :  $\overline{\neg p} = p$   $\overline{p} = \neg p$

## Semantics:

- A partial interpretation  $I$  is a consistent set of literals ( $x \notin I$  or  $\bar{x} \notin I$ )  
It may falsify or satisfy a formula or neither (because it is partial)
- $I$  satisfies a CNF formula  $\phi$  iff  $I$  satisfies all clauses  $c \in \phi$   
 $I$  falsifies a CNF formula  $\phi$  iff  $I$  falsifies some clause  $c \in \phi$
- $I$  satisfies a clause  $c$  iff  $I$  satisfies some literal  $x \in c$   
 $I$  falsifies a clause  $c$  iff  $I$  falsifies all literals  $x \in c$
- $I$  satisfies a literal  $x$  iff  $x \in I$   
 $I$  falsifies a literal  $x$  iff  $\bar{x} \in I$

## SAT Algorithm 1a: Nondeterministic

Let  $I = \{\}$ . Repeat:

1. If  $I$  falsifies some  $c \in \phi$ : return NO
2. Select a variable  $x$  such that  $x, \bar{x} \notin I$
3. If there is none: return YES
4. Let  $I = I \cup \{x\}$  or  $I = I \cup \{\neg x\}$

# SAT Algorithm 1a: Nondeterministic

Let  $I = \{\}$ . Repeat:

1. If  $I$  falsifies some  $c \in \phi$ : return NO
2. Select a variable  $x$  such that  $x, \bar{x} \notin I$
3. If there is none: return YES
4. Let  $I = I \cup \{x\}$  **or**  $I = I \cup \{\neg x\}$

Bad news: Search space is exponential in number of variables.  
(It is believed that) we can't do better.

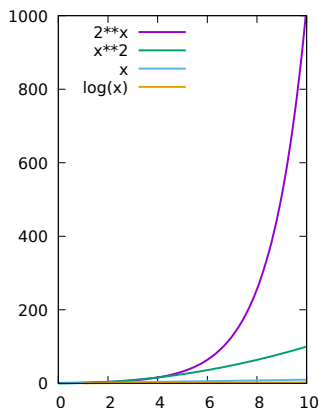
# Complexity Theory: Big O Notation

- Complexity is (usually) measured in the length  $n$  of the input:

- ▶  $f(n) = \mathcal{O}(g(n))$  iff for some  $k$ , for large  $n$ ,  $f(n) \leq k \cdot g(n)$
- ▶ Exponential complexity:  $\mathcal{O}(k^n)$
- ▶ Polynomial complexity:  $\mathcal{O}(n^k)$
- ▶ Linear complexity:  $\mathcal{O}(n)$
- ▶ Logarithmic complexity:  $\mathcal{O}(\log n)$
- ▶ Constant complexity:  $\mathcal{O}(1)$

- Length of input = number of symbols:

- ▶ Length of " $\neg(p \wedge q)$ " = 6
- ▶ Length of 173 in decimal = 3
- ▶ Length of 173 in binary = 8



- Time complexity: number of computation steps.
- Space complexity: amount of memory used.
- Time is upper bound for space.



# Complexity Theory: P and NP

## Definition: decision problem, complexity class

A **decision problem** is a yes/no question over a set of instances.

An **instance** is a finite sequence of symbols from finite alphabet.

A **complexity class** is a set of decision problems of related complexity.

# Complexity Theory: P and NP

## Definition: decision problem, complexity class

A **decision problem** is a yes/no question over a set of instances.  
An **instance** is a finite sequence of symbols from finite alphabet.  
A **complexity class** is a set of decision problems of related complexity.

## Definition: complexity class P, NP

$A \in P$  iff solvable in poly. time by a deterministic machine.  
 $A \in NP$  iff solvable in poly. time by a nondeterministic machine.

- Det. machine takes a predetermined action in each state.
- Nondet. m. takes **one out of a set of possible actions** in each state.
  - ▶ The machine **guesses** which **action** to take.
  - ▶ The machine guesses the shortest way to **"yes"** if there is one.

# Complexity Theory: P and NP

## Definition: decision problem, complexity class

A **decision problem** is a yes/no question over a set of instances.  
An **instance** is a finite sequence of symbols from finite alphabet.  
A **complexity class** is a set of decision problems of related complexity.

## Definition: complexity class P, NP

$A \in P$  iff solvable in poly. time by a deterministic machine.  
 $A \in NP$  iff solvable in poly. time by a nondeterministic machine.

- Det. machine takes a predetermined action in each state.
- Nondet. m. takes **one out of a set of possible actions** in each state.
  - ▶ The machine **guesses** which **action** to take.
  - ▶ The machine guesses the shortest way to **"yes"** if there is one.

## Theorem: alternative characterisation of NP

$A \in NP$  iff "yes"-proof verifiable by a det. machine in poly. time.

# Complexity Theory: NP-Completeness

A problem is NP-complete if it's among the hardest problems in NP.

# Complexity Theory: NP-Completeness

A problem is NP-complete if it's among the hardest problems in NP.

## Definition: reduction from $A$ to $B$

$A \leq_P B$  iff for some function  $f$  from  $A$ -instances to  $B$ -instances:  
for all instances of  $x$  of  $A$ :  
 $f(x)$  is computable in polynomial time and  
 $x$  is a "yes"-instance of  $A$  iff  $f(x)$  is a "yes"-instance of  $B$ .

# Complexity Theory: NP-Completeness

A problem is NP-complete if it's among the hardest problems in NP.

## Definition: reduction from $A$ to $B$

$A \leq_P B$  iff for some function  $f$  from  $A$ -instances to  $B$ -instances:  
for all instances of  $x$  of  $A$ :  
 $f(x)$  is computable in polynomial time and  
 $x$  is a “yes”-instance of  $A$  iff  $f(x)$  is a “yes”-instance of  $B$ .

## Definition: NP-hard, NP-complete

$B$  is NP-hard      iff  $A \leq_P B$  for all  $A \in \text{NP}$ .

$B$  is NP-complete iff  $B \in \text{NP}$  and  $B$  is NP-hard.

# Complexity Theory: NP-Completeness

A problem is NP-complete if it's among the hardest problems in NP.

## Definition: reduction from $A$ to $B$

$A \leq_P B$  iff for some function  $f$  from  $A$ -instances to  $B$ -instances:  
for all instances of  $x$  of  $A$ :  
 $f(x)$  is computable in polynomial time and  
 $x$  is a “yes”-instance of  $A$  iff  $f(x)$  is a “yes”-instance of  $B$ .

## Definition: NP-hard, NP-complete

$B$  is NP-hard      iff  $A \leq_P B$  for all  $A \in \text{NP}$ .  
 $B$  is NP-complete iff  $B \in \text{NP}$  and  $B$  is NP-hard.

## Theorem: complexity of SAT

SAT and 3-SAT are NP-complete.

# Complexity Theory: NP-Completeness

A problem is NP-complete if it's among the hardest problems in NP.

## Definition: reduction from $A$ to $B$

$A \leq_P B$  iff for some function  $f$  from  $A$ -instances to  $B$ -instances:  
for all instances of  $x$  of  $A$ :  
 $f(x)$  is computable in polynomial time and  
 $x$  is a "yes"-instance of  $A$  iff  $f(x)$  is a "yes"-instance of  $B$ .

## Definition: NP-hard, NP-complete

$B$  is NP-hard      iff  $A \leq_P B$  for all  $A \in \text{NP}$ .  
 $B$  is NP-complete iff  $B \in \text{NP}$  and  $B$  is NP-hard.

## Theorem: complexity of SAT

SAT and 3-SAT are NP-complete.

Common hypothesis:  $P \neq \text{NP}$ . (Thus  $\text{SAT} \notin P$ .)



# Motivation

NP-complete problems ...

- ...include many real-world problems.
- ...are believed to require exponential time (bad).
- ...can be reduced to SAT in polynomial time (good).

# Motivation

NP-complete problems ...

- ...include many real-world problems.
- ...are believed to require exponential time (bad).
- ...can be reduced to SAT in polynomial time (good).

Modern SAT solvers are very fast for many real-world instances:

- Hardware verification
- Software verification
- Planning
- Cryptography
- Answer Set solving

# SAT Algorithm 1b: Deterministic

Let  $I = \{\}$ . Repeat:

1. If  $I$  falsifies some  $c \in \phi$ :

1.1 Backtrack to the last decision  $\neg x$

undo last decisions

1.2 If there is none: return NO

1.3 Let  $I = I \cup \{x\}$

flip last negative decision

2. Else:

2.1 Select a variable  $x$  such that  $x, \bar{x} \notin I$

2.2 If there is none: return YES

2.3 Let  $I = I \cup \{\neg x\}$

make a decision

# Outline

Satisfiability and Complexity Theory

Solver Ingredient 1: Watched-Literal Scheme

Solver Ingredient 2: Conflict-Driven Clause Learning

Other Solver Ingredients

Conclusion

Beyond SAT

## Unit Propagation: Idea

Unit resolution rule: from  $p$  and  $q$  and  $(\neg p \vee \neg q \vee r)$  infer  $r$ .

## Unit Propagation: Idea

Unit resolution rule: from  $p$  and  $q$  and  $(\neg p \vee \neg q \vee r)$  infer  $r$ .

In SAT, we want to find a satisfying assignment  $I$  of  $\phi$ :

If  $p, q \in I$ , then  $I$  can satisfy  $(\neg p \vee \neg q \vee r) \in \phi$  only if  $r \in I$ .

# Unit Propagation

Definition: closure of  $I$  under unit propagation relative to  $\phi$

- Let  $I^0 = I$
- Repeat for  $j > 0$  until  $I^j = I^{j+1}$ :
  - ▶ If there is a  $(x_1 \vee \dots \vee x_k) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Return **conflict**  $(x_1 \vee \dots \vee x_k)$
  - ▶ If there is a  $(x_1 \vee \dots \vee x_{k+1}) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Let  $I^{j+1} = I^j \cup \{x_{k+1}\}$
- Return  $I^j$

# Unit Propagation

Definition: closure of  $I$  under unit propagation relative to  $\phi$

- Let  $I^0 = I$
- Repeat for  $j > 0$  until  $I^j = I^{j+1}$ :
  - ▶ If there is a  $(x_1 \vee \dots \vee x_k) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Return **conflict**  $(x_1 \vee \dots \vee x_k)$
  - ▶ If there is a  $(x_1 \vee \dots \vee x_{k+1}) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Let  $I^{j+1} = I^j \cup \{x_{k+1}\}$
- Return  $I^j$

Ex. 1:  $I = \{\neg p\}$     $\phi = \{(\neg p \vee \neg q \vee s), (p \vee \neg q \vee r), (p \vee q)\}$



# Unit Propagation

Definition: closure of  $I$  under unit propagation relative to  $\phi$

- Let  $I^0 = I$
- Repeat for  $j > 0$  until  $I^j = I^{j+1}$ :
  - ▶ If there is a  $(x_1 \vee \dots \vee x_k) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Return **conflict**  $(x_1 \vee \dots \vee x_k)$
  - ▶ If there is a  $(x_1 \vee \dots \vee x_{k+1}) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Let  $I^{j+1} = I^j \cup \{x_{k+1}\}$
- Return  $I^j$

Ex. 1:  $I = \{\neg p\}$     $\phi = \{(\neg p \vee \neg q \vee s), (p \vee \neg q \vee r), (p \vee q)\}$

- $I^0 = \{\neg p\}$

# Unit Propagation

Definition: closure of  $I$  under unit propagation relative to  $\phi$

- Let  $I^0 = I$
- Repeat for  $j > 0$  until  $I^j = I^{j+1}$ :
  - ▶ If there is a  $(x_1 \vee \dots \vee x_k) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Return **conflict**  $(x_1 \vee \dots \vee x_k)$
  - ▶ If there is a  $(x_1 \vee \dots \vee x_{k+1}) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Let  $I^{j+1} = I^j \cup \{x_{k+1}\}$
- Return  $I^j$

Ex. 1:  $I = \{\neg p\}$     $\phi = \{(\neg p \vee \neg q \vee s), (p \vee \neg q \vee r), (p \vee q)\}$

- $I^0 = \{\neg p\}$

# Unit Propagation

Definition: closure of  $I$  under unit propagation relative to  $\phi$

- Let  $I^0 = I$
- Repeat for  $j > 0$  until  $I^j = I^{j+1}$ :
  - ▶ If there is a  $(x_1 \vee \dots \vee x_k) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Return **conflict**  $(x_1 \vee \dots \vee x_k)$
  - ▶ If there is a  $(x_1 \vee \dots \vee x_{k+1}) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Let  $I^{j+1} = I^j \cup \{x_{k+1}\}$
- Return  $I^j$

Ex. 1:  $I = \{\neg p\}$     $\phi = \{(\neg p \vee \neg q \vee s), (p \vee \neg q \vee r), (p \vee q)\}$

- $I^0 = \{\neg p\}$
- $I^1 = \{\neg p, q\}$

# Unit Propagation

Definition: closure of  $I$  under unit propagation relative to  $\phi$

- Let  $I^0 = I$
- Repeat for  $j > 0$  until  $I^j = I^{j+1}$ :
  - ▶ If there is a  $(x_1 \vee \dots \vee x_k) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Return **conflict**  $(x_1 \vee \dots \vee x_k)$
  - ▶ If there is a  $(x_1 \vee \dots \vee x_{k+1}) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Let  $I^{j+1} = I^j \cup \{x_{k+1}\}$
- Return  $I^j$

Ex. 1:  $I = \{\neg p\}$     $\phi = \{(\neg p \vee \neg q \vee s), (p \vee \neg q \vee r), (p \vee q)\}$

- $I^0 = \{\neg p\}$
- $I^1 = \{\neg p, q\}$

# Unit Propagation

Definition: closure of  $I$  under unit propagation relative to  $\phi$

- Let  $I^0 = I$
- Repeat for  $j > 0$  until  $I^j = I^{j+1}$ :
  - ▶ If there is a  $(x_1 \vee \dots \vee x_k) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Return **conflict**  $(x_1 \vee \dots \vee x_k)$
  - ▶ If there is a  $(x_1 \vee \dots \vee x_{k+1}) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Let  $I^{j+1} = I^j \cup \{x_{k+1}\}$
- Return  $I^j$

Ex. 1:  $I = \{\neg p\}$     $\phi = \{(\neg p \vee \neg q \vee s), (p \vee \neg q \vee r), (p \vee q)\}$

- $I^0 = \{\neg p\}$
- $I^1 = \{\neg p, q\}$
- $I^2 = \{\neg p, q, r\}$

# Unit Propagation

Definition: closure of  $I$  under unit propagation relative to  $\phi$

- Let  $I^0 = I$
- Repeat for  $j > 0$  until  $I^j = I^{j+1}$ :
  - ▶ If there is a  $(x_1 \vee \dots \vee x_k) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Return **conflict**  $(x_1 \vee \dots \vee x_k)$
  - ▶ If there is a  $(x_1 \vee \dots \vee x_{k+1}) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Let  $I^{j+1} = I^j \cup \{x_{k+1}\}$
- Return  $I^j$

Ex. 1:  $I = \{\neg p\}$     $\phi = \{(\neg p \vee \neg q \vee s), (p \vee \neg q \vee r), (p \vee q)\}$

- $I^0 = \{\neg p\}$
- $I^1 = \{\neg p, q\}$
- $I^2 = \{\neg p, q, r\}$

# Unit Propagation

Definition: closure of  $I$  under unit propagation relative to  $\phi$

- Let  $I^0 = I$
- Repeat for  $j > 0$  until  $I^j = I^{j+1}$ :
  - ▶ If there is a  $(x_1 \vee \dots \vee x_k) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Return **conflict**  $(x_1 \vee \dots \vee x_k)$
  - ▶ If there is a  $(x_1 \vee \dots \vee x_{k+1}) \in \phi$  with  $\bar{x}_1, \dots, \bar{x}_k \in I^j$ :  
Let  $I^{j+1} = I^j \cup \{x_{k+1}\}$
- Return  $I^j$

Ex. 2:  $I = \{\neg p\}$     $\phi = \{(\neg p \vee \neg q \vee s), (p \vee \neg q), (p \vee q)\}$

- $I^0 = \{\neg p\}$
- $I^1 = \{\neg p, q\}$
- Conflict with  $(p \vee \neg q)$

## SAT Algorithm 2: Adding Unit Propagation

Let  $I = \{\}$ . Repeat:

1. Close  $I$  under unit propagation relative to  $\phi$
2. If conflict:
  - 2.1 Backtrack to the last decision  $\neg x$  undo last decisions
  - 2.2 If there is none: return NO
  - 2.3 Let  $I = I \cup \{x\}$  flip last negative decision
3. Else:
  - 3.1 Select a variable  $x$  such that  $x, \bar{x} \notin I$
  - 3.2 If there is none: return YES
  - 3.3 Let  $I = I \cup \{\neg x\}$  make a decision



# Watched-Literal Scheme: Motivation

- Algorithm 2 spends almost all its time on
  - ▶ unit propagation and
  - ▶ backtracking.
- Watched-Literal Scheme is a **lazy data structure** for
  - ▶ fast unit propagation and
  - ▶ very cheap backtracking.

## Watched-Literal Scheme

- Recall: (i) If  $(x_1 \vee \dots \vee x_k) \in \phi$  and  $\bar{x}_1, \dots, \bar{x}_k \in I$ : Return conflict.  
(ii) If  $(x_1 \vee \dots \vee x_{k+1}) \in \phi$  and  $\bar{x}_1, \dots, \bar{x}_k \in I$ : Add  $x_{k+1}$  to  $I$ .

How to close  $I$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$ :

- (i) If all  $\bar{x}_i \in I$ : Return conflict.
- (ii) If all but one  $\bar{x}_i \in I$ : Add remaining  $x_j$  to  $I$ .

Ex.:  $\phi = (p \vee q \vee r \vee s) \wedge (p \vee q \vee t) \wedge (u \vee v \vee w)$   
 $I = \{\}$

## Watched-Literal Scheme

- Recall: (i) If  $(x_1 \vee \dots \vee x_k) \in \phi$  and  $\bar{x}_1, \dots, \bar{x}_k \in I$ : Return conflict.  
(ii) If  $(x_1 \vee \dots \vee x_{k+1}) \in \phi$  and  $\bar{x}_1, \dots, \bar{x}_k \in I$ : Add  $x_{k+1}$  to  $I$ .

How to close  $I$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$ :

- (i) If all  $\bar{x}_i \in I$ : Return conflict.
- (ii) If all but one  $\bar{x}_i \in I$ : Add remaining  $x_j$  to  $I$ .

Ex.:  $\phi = (p \vee q \vee r \vee s) \wedge (p \vee q \vee t) \wedge (u \vee v \vee w)$   
 $I = \{\neg p\}$

## Watched-Literal Scheme

- Recall: (i) If  $(x_1 \vee \dots \vee x_k) \in \phi$  and  $\bar{x}_1, \dots, \bar{x}_k \in I$ : Return conflict.  
(ii) If  $(x_1 \vee \dots \vee x_{k+1}) \in \phi$  and  $\bar{x}_1, \dots, \bar{x}_k \in I$ : Add  $x_{k+1}$  to  $I$ .

How to close  $I$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$ :

- (i) If all  $\bar{x}_i \in I$ : Return conflict.
- (ii) If all but one  $\bar{x}_i \in I$ : Add remaining  $x_j$  to  $I$ .

Ex.:  $\phi = (p \vee q \vee r \vee s) \wedge (p \vee q \vee t) \wedge (u \vee v \vee w)$   
 $I = \{\neg p, \neg q\}$

## Watched-Literal Scheme

- Recall: (i) If  $(x_1 \vee \dots \vee x_k) \in \phi$  and  $\bar{x}_1, \dots, \bar{x}_k \in I$ : Return conflict.  
(ii) If  $(x_1 \vee \dots \vee x_{k+1}) \in \phi$  and  $\bar{x}_1, \dots, \bar{x}_k \in I$ : Add  $x_{k+1}$  to  $I$ .

How to close  $I$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$ :

- (i) If all  $\bar{x}_i \in I$ : Return conflict.
- (ii) If all but one  $\bar{x}_i \in I$ : Add remaining  $x_j$  to  $I$ .

Ex.:  $\phi = (p \vee q \vee r \vee s) \wedge (p \vee q \vee t) \wedge (u \vee v \vee w)$   
 $I = \{\neg p, \neg q, t\}$

## Watched-Literal Scheme

- Recall: (i) If  $(x_1 \vee \dots \vee x_k) \in \phi$  and  $\bar{x}_1, \dots, \bar{x}_k \in I$ : Return conflict.  
(ii) If  $(x_1 \vee \dots \vee x_{k+1}) \in \phi$  and  $\bar{x}_1, \dots, \bar{x}_k \in I$ : Add  $x_{k+1}$  to  $I$ .

How to close  $I$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$ :

- (i) If all  $\bar{x}_i \in I$ : Return conflict.
- (ii) If all but one  $\bar{x}_i \in I$ : Add remaining  $x_j$  to  $I$ .

Ex.:  $\phi = (p \vee q \vee r \vee s) \wedge (p \vee q \vee t) \wedge (u \vee v \vee w)$   
 $I = \{\}$

## Watched-Literal Scheme

For every  $c \in \phi$ , select two distinct **watched literals**  $x_c, y_c \in c$   
such that  $\bar{x}_c \notin I$  if possible, otherwise choose arbitrarily  
and  $\bar{y}_c \notin I$  if possible, otherwise choose arbitrarily.

How to close  $I$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$ :

- (i) If all  $\bar{x}_i \in I$ : Return conflict.
- (ii) If all but one  $\bar{x}_i \in I$ : Add remaining  $x_j$  to  $I$ .

Ex.:  $\phi = (\underline{p} \vee \underline{q} \vee r \vee s) \wedge (\underline{p} \vee \underline{q} \vee t) \wedge (\underline{u} \vee \underline{v} \vee w)$   
 $I = \{\}$

## Watched-Literal Scheme

For every  $c \in \phi$ , select two distinct **watched literals**  $x_c, y_c \in c$   
such that  $\bar{x}_c \notin I$  if possible, otherwise choose arbitrarily  
and  $\bar{y}_c \notin I$  if possible, otherwise choose arbitrarily.

How to close  $I$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$  with watched literals  $x_c, y_c$ :

- (i) If  $\bar{x}_c \in I, \bar{y}_c \in I$ : Try to update  $x_c, y_c$ . Otherwise return conflict.
- (ii) If  $\bar{x}_c \in I, \bar{y}_c \notin I$ : Try to update  $x_c$ . Otherwise add  $y_c$  to  $I$ .  
If  $\bar{x}_c \notin I, \bar{y}_c \in I$ : Try to update  $y_c$ . Otherwise add  $x_c$  to  $I$ .

Ex.:  $\phi = (\underline{p} \vee \underline{q} \vee r \vee s) \wedge (\underline{p} \vee \underline{q} \vee t) \wedge (\underline{u} \vee \underline{v} \vee w)$   
 $I = \{\}$



## Watched-Literal Scheme

For every  $c \in \phi$ , select two distinct **watched literals**  $x_c, y_c \in c$   
such that  $\bar{x}_c \notin I$  if possible, otherwise choose arbitrarily  
and  $\bar{y}_c \notin I$  if possible, otherwise choose arbitrarily.

How to close  $I$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$  with watched literals  $x_c, y_c$ :

- (i) If  $\bar{x}_c \in I, \bar{y}_c \in I$ : Try to update  $x_c, y_c$ . Otherwise return conflict.
- (ii) If  $\bar{x}_c \in I, \bar{y}_c \notin I$ : Try to update  $x_c$ . Otherwise add  $y_c$  to  $I$ .  
If  $\bar{x}_c \notin I, \bar{y}_c \in I$ : Try to update  $y_c$ . Otherwise add  $x_c$  to  $I$ .

Ex.:  $\phi = (\underline{p} \vee \underline{q} \vee r \vee s) \wedge (\underline{p} \vee \underline{q} \vee t) \wedge (\underline{u} \vee \underline{v} \vee w)$   
 $I = \{\neg p\}$

## Watched-Literal Scheme

For every  $c \in \phi$ , select two distinct **watched literals**  $x_c, y_c \in c$   
such that  $\bar{x}_c \notin I$  if possible, otherwise choose arbitrarily  
and  $\bar{y}_c \notin I$  if possible, otherwise choose arbitrarily.

How to close  $I$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$  with watched literals  $x_c, y_c$ :

- (i) If  $\bar{x}_c \in I, \bar{y}_c \in I$ : Try to update  $x_c, y_c$ . Otherwise return conflict.
- (ii) If  $\bar{x}_c \in I, \bar{y}_c \notin I$ : Try to update  $x_c$ . Otherwise add  $y_c$  to  $I$ .  
If  $\bar{x}_c \notin I, \bar{y}_c \in I$ : Try to update  $y_c$ . Otherwise add  $x_c$  to  $I$ .

Ex.:  $\phi = (p \vee \underline{q} \vee \underline{r} \vee s) \wedge (p \vee \underline{q} \vee \underline{t}) \wedge (\underline{u} \vee \underline{v} \vee w)$   
 $I = \{\neg p\}$

## Watched-Literal Scheme

For every  $c \in \phi$ , select two distinct **watched literals**  $x_c, y_c \in c$   
such that  $\bar{x}_c \notin I$  if possible, otherwise choose arbitrarily  
and  $\bar{y}_c \notin I$  if possible, otherwise choose arbitrarily.

How to close  $I$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$  with watched literals  $x_c, y_c$ :

- (i) If  $\bar{x}_c \in I, \bar{y}_c \in I$ : Try to update  $x_c, y_c$ . Otherwise return conflict.
- (ii) If  $\bar{x}_c \in I, \bar{y}_c \notin I$ : Try to update  $x_c$ . Otherwise add  $y_c$  to  $I$ .  
If  $\bar{x}_c \notin I, \bar{y}_c \in I$ : Try to update  $y_c$ . Otherwise add  $x_c$  to  $I$ .

Ex.:  $\phi = (p \vee \underline{q} \vee \underline{r} \vee s) \wedge (p \vee \underline{q} \vee \underline{t}) \wedge (\underline{u} \vee \underline{v} \vee w)$   
 $I = \{\neg p, \neg q\}$

## Watched-Literal Scheme

For every  $c \in \phi$ , select two distinct **watched literals**  $x_c, y_c \in c$   
such that  $\bar{x}_c \notin I$  if possible, otherwise choose arbitrarily  
and  $\bar{y}_c \notin I$  if possible, otherwise choose arbitrarily.

How to close  $I$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$  with watched literals  $x_c, y_c$ :

- (i) If  $\bar{x}_c \in I, \bar{y}_c \in I$ : Try to update  $x_c, y_c$ . Otherwise return conflict.
- (ii) If  $\bar{x}_c \in I, \bar{y}_c \notin I$ : Try to update  $x_c$ . Otherwise add  $y_c$  to  $I$ .  
If  $\bar{x}_c \notin I, \bar{y}_c \in I$ : Try to update  $y_c$ . Otherwise add  $x_c$  to  $I$ .

Ex.:  $\phi = (p \vee q \vee \underline{r} \vee \underline{s}) \wedge (p \vee \underline{q} \vee \underline{t}) \wedge (\underline{u} \vee \underline{v} \vee w)$   
 $I = \{\neg p, \neg q\}$

## Watched-Literal Scheme

For every  $c \in \phi$ , select two distinct **watched literals**  $x_c, y_c \in c$   
such that  $\bar{x}_c \notin I$  if possible, otherwise choose arbitrarily  
and  $\bar{y}_c \notin I$  if possible, otherwise choose arbitrarily.

How to close  $I$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$  with watched literals  $x_c, y_c$ :

- (i) If  $\bar{x}_c \in I, \bar{y}_c \in I$ : Try to update  $x_c, y_c$ . Otherwise return conflict.
- (ii) If  $\bar{x}_c \in I, \bar{y}_c \notin I$ : Try to update  $x_c$ . Otherwise add  $y_c$  to  $I$ .  
If  $\bar{x}_c \notin I, \bar{y}_c \in I$ : Try to update  $y_c$ . Otherwise add  $x_c$  to  $I$ .

Ex.:  $\phi = (p \vee q \vee \underline{r} \vee \underline{s}) \wedge (p \vee \underline{q} \vee \underline{t}) \wedge (\underline{u} \vee \underline{v} \vee w)$   
 $I = \{\neg p, \neg q, t\}$

## Watched-Literal Scheme

For every  $c \in \phi$ , select two distinct **watched literals**  $x_c, y_c \in c$   
such that  $\bar{x}_c \notin I$  if possible, otherwise choose arbitrarily  
and  $\bar{y}_c \notin I$  if possible, otherwise choose arbitrarily.

Suppose  $I$  is closed under UP relative to  $\phi$ .

How to close  $I' = I \cup \{z\}$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$  that watches  $\bar{z} = x_c$  (w.l.o.g.):

1. Try to update  $x_c$ .
2. Otherwise: If  $\bar{y}_c \in I'$ : Return conflict.  
If  $\bar{y}_c \notin I'$ : Add  $y_c$  to  $I'$ .

Ex.:  $\phi = (\underline{p} \vee \underline{q} \vee r \vee s) \wedge (\underline{p} \vee \underline{q} \vee t) \wedge (\underline{u} \vee \underline{v} \vee w)$   
 $I = \{\}$

## Watched-Literal Scheme

For every  $c \in \phi$ , select two distinct **watched literals**  $x_c, y_c \in c$   
such that  $\bar{x}_c \notin I$  if possible, otherwise choose arbitrarily  
and  $\bar{y}_c \notin I$  if possible, otherwise choose arbitrarily.

Suppose  $I$  is closed under UP relative to  $\phi$ .

How to close  $I' = I \cup \{z\}$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$  that watches  $\bar{z} = x_c$  (w.l.o.g.):

1. Try to update  $x_c$ .
2. Otherwise: If  $\bar{y}_c \in I'$ : Return conflict.  
If  $\bar{y}_c \notin I'$ : Add  $y_c$  to  $I'$ .

Ex.:  $\phi = (\underline{p} \vee \underline{q} \vee r \vee s) \wedge (\underline{p} \vee \underline{q} \vee t) \wedge (\underline{u} \vee \underline{v} \vee w)$   
 $I = \{\neg p\}$

## Watched-Literal Scheme

For every  $c \in \phi$ , select two distinct **watched literals**  $x_c, y_c \in c$   
such that  $\bar{x}_c \notin I$  if possible, otherwise choose arbitrarily  
and  $\bar{y}_c \notin I$  if possible, otherwise choose arbitrarily.

Suppose  $I$  is closed under UP relative to  $\phi$ .

How to close  $I' = I \cup \{z\}$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$  that watches  $\bar{z} = x_c$  (w.l.o.g.):

1. Try to update  $x_c$ .
2. Otherwise: If  $\bar{y}_c \in I'$ : Return conflict.  
If  $\bar{y}_c \notin I'$ : Add  $y_c$  to  $I'$ .

Ex.:  $\phi = (p \vee \underline{q} \vee \underline{r} \vee s) \wedge (p \vee \underline{q} \vee \underline{t}) \wedge (\underline{u} \vee \underline{v} \vee w)$   
 $I = \{\neg p\}$



## Watched-Literal Scheme

For every  $c \in \phi$ , select two distinct **watched literals**  $x_c, y_c \in c$   
such that  $\bar{x}_c \notin I$  if possible, otherwise choose arbitrarily  
and  $\bar{y}_c \notin I$  if possible, otherwise choose arbitrarily.

Suppose  $I$  is closed under UP relative to  $\phi$ .

How to close  $I' = I \cup \{z\}$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$  that watches  $\bar{z} = x_c$  (w.l.o.g.):

1. Try to update  $x_c$ .
2. Otherwise: If  $\bar{y}_c \in I'$ : Return conflict.  
If  $\bar{y}_c \notin I'$ : Add  $y_c$  to  $I'$ .

Ex.:  $\phi = (p \vee \underline{q} \vee \underline{r} \vee s) \wedge (p \vee \underline{q} \vee \underline{t}) \wedge (\underline{u} \vee \underline{v} \vee w)$   
 $I = \{\neg p, \neg q\}$

## Watched-Literal Scheme

For every  $c \in \phi$ , select two distinct **watched literals**  $x_c, y_c \in c$   
such that  $\bar{x}_c \notin I$  if possible, otherwise choose arbitrarily  
and  $\bar{y}_c \notin I$  if possible, otherwise choose arbitrarily.

Suppose  $I$  is closed under UP relative to  $\phi$ .

How to close  $I' = I \cup \{z\}$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$  that watches  $\bar{z} = x_c$  (w.l.o.g.):

1. Try to update  $x_c$ .
2. Otherwise: If  $\bar{y}_c \in I'$ : Return conflict.  
If  $\bar{y}_c \notin I'$ : Add  $y_c$  to  $I'$ .

Ex.:  $\phi = (p \vee q \vee \underline{r} \vee \underline{s}) \wedge (p \vee \underline{q} \vee \underline{t}) \wedge (\underline{u} \vee \underline{v} \vee w)$   
 $I = \{\neg p, \neg q\}$

## Watched-Literal Scheme

For every  $c \in \phi$ , select two distinct **watched literals**  $x_c, y_c \in c$   
such that  $\bar{x}_c \notin I$  if possible, otherwise choose arbitrarily  
and  $\bar{y}_c \notin I$  if possible, otherwise choose arbitrarily.

Suppose  $I$  is closed under UP relative to  $\phi$ .

How to close  $I' = I \cup \{z\}$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$  that watches  $\bar{z} = x_c$  (w.l.o.g.):

1. Try to update  $x_c$ .
2. Otherwise: If  $\bar{y}_c \in I'$ : Return conflict.  
If  $\bar{y}_c \notin I'$ : Add  $y_c$  to  $I'$ .

Ex.:  $\phi = (p \vee q \vee \underline{r} \vee \underline{s}) \wedge (p \vee \underline{q} \vee \underline{t}) \wedge (\underline{u} \vee \underline{v} \vee w)$   
 $I = \{\neg p, \neg q, t\}$

## Watched-Literal Scheme

For every  $c \in \phi$ , select two distinct **watched literals**  $x_c, y_c \in c$   
such that  $\bar{x}_c \notin I$  if possible, otherwise choose arbitrarily  
and  $\bar{y}_c \notin I$  if possible, otherwise choose arbitrarily.

Suppose  $I$  is closed under UP relative to  $\phi$ .

How to close  $I' = I \cup \{z_1, \dots, z_\ell\}$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$  that watches  $\bar{z}_1 = x_c$  (w.l.o.g.):

1. Try to update  $x_c$ .
2. Otherwise: If  $\bar{y}_c \in I'$ : Return conflict.  
If  $\bar{y}_c \notin I'$ : Add  $y_c$  to  $I'$ .

Now  $I \cup \{z_1\}$  is closed under UP relative to  $\phi$ . Repeat for  $z_2$ , and so on.

Ex.:  $\phi = (p \vee q \vee \underline{r} \vee \underline{s}) \wedge (p \vee \underline{q} \vee \underline{t}) \wedge (\underline{u} \vee \underline{v} \vee w)$   
 $I = \{\neg p, \neg q, t\}$

## Watched-Literal Scheme

For every  $c \in \phi$ , select two distinct **watched literals**  $x_c, y_c \in c$   
such that  $\bar{x}_c \notin I$  if possible, otherwise choose arbitrarily  
and  $\bar{y}_c \notin I$  if possible, otherwise choose arbitrarily.

Suppose  $I$  is closed under UP relative to  $\phi$ .

How to close  $I' = I \cup \{z_1, \dots, z_\ell\}$  under UP relative to  $\phi$ ?

For every  $(x_1 \vee \dots \vee x_k) \in \phi$  that watches  $\bar{z}_1 = x_c$  (w.l.o.g.):

1. Try to update  $x_c$ .
2. Otherwise: If  $\bar{y}_c \in I'$ : Return conflict.  
If  $\bar{y}_c \notin I'$ : Add  $y_c$  to  $I'$ .

Now  $I \cup \{z_1\}$  is closed under UP relative to  $\phi$ . Repeat for  $z_2$ , and so on.

Ex.:  $\phi = (p \vee q \vee \underline{r} \vee \underline{s}) \wedge (p \vee \underline{q} \vee \underline{t}) \wedge (\underline{u} \vee \underline{v} \vee w)$   
 $I = \{\neg p, \neg q, t\}$

How to backtrack? Remove literals from  $I$ . No update of  $x_c, y_c$  needed!

## Watched-Literal Scheme: Example

Clauses:       $p \vee q \vee s$                        $\neg r \vee \neg s \vee \neg t$   
                  $t \vee \neg u$                        $t \vee \neg v$                        $t \vee \neg w$   
                  $u \vee v \vee w \vee y$                $v \vee \neg y$

Decisions:     $\neg p, \neg q, r$

## Watched-Literal Scheme: Example

$I$	Clauses and Watched Literals						
	$p \vee q \vee s$	$\bar{r} \vee \bar{s} \vee \bar{t}$	$t \vee \bar{u}$	$t \vee \bar{v}$	$t \vee \bar{w}$	$u \vee v \vee w \vee y$	$v \vee \bar{y}$
	$p, q$	$\bar{r}, \bar{s}$	$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$	$u, v$	$v, \bar{y}$

Recall watched-literal invariant:  $\bar{x}_c, \bar{y}_c \notin I$  if possible.

## Watched-Literal Scheme: Example

$I$	Clauses and Watched Literals						
	$p \vee q \vee s$	$\bar{r} \vee \bar{s} \vee \bar{t}$	$t \vee \bar{u}$	$t \vee \bar{v}$	$t \vee \bar{w}$	$u \vee v \vee w \vee y$	$v \vee \bar{y}$
	$p, q$	$\bar{r}, \bar{s}$	$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$	$u, v$	$v, \bar{y}$
$\bar{p}$							

Recall watched-literal invariant:  $\bar{x}_c, \bar{y}_c \notin I$  if possible.



## Watched-Literal Scheme: Example

$I$	Clauses and Watched Literals						
	$p \vee q \vee s$	$\bar{r} \vee \bar{s} \vee \bar{t}$	$t \vee \bar{u}$	$t \vee \bar{v}$	$t \vee \bar{w}$	$u \vee v \vee w \vee y$	$v \vee \bar{y}$
	$p, q$	$\bar{r}, \bar{s}$	$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$	$u, v$	$v, \bar{y}$
$\bar{p}$	$s, q$						

Recall watched-literal invariant:  $\bar{x}_c, \bar{y}_c \notin I$  if possible.

## Watched-Literal Scheme: Example

$I$	Clauses and Watched Literals						
	$p \vee q \vee s$	$\bar{r} \vee \bar{s} \vee \bar{t}$	$t \vee \bar{u}$	$t \vee \bar{v}$	$t \vee \bar{w}$	$u \vee v \vee w \vee y$	$v \vee \bar{y}$
	$p, q$	$\bar{r}, \bar{s}$	$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$	$u, v$	$v, \bar{y}$
$\bar{p}$	$s, q$						
$\bar{q}$							

Recall watched-literal invariant:  $\bar{x}_c, \bar{y}_c \notin I$  if possible.

# Watched-Literal Scheme: Example

$I$	Clauses and Watched Literals						
	$p \vee q \vee s$	$\bar{r} \vee \bar{s} \vee \bar{t}$	$t \vee \bar{u}$	$t \vee \bar{v}$	$t \vee \bar{w}$	$u \vee v \vee w \vee y$	$v \vee \bar{y}$
	$p, q$	$\bar{r}, \bar{s}$	$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$	$u, v$	$v, \bar{y}$
$\bar{p}$	$s, q$						
$\bar{q}$	$s, q$						
$s$							

Recall watched-literal invariant:  $\bar{x}_c, \bar{y}_c \notin I$  if possible.

# Watched-Literal Scheme: Example

$I$	Clauses and Watched Literals						
	$p \vee q \vee s$	$\bar{r} \vee \bar{s} \vee \bar{t}$	$t \vee \bar{u}$	$t \vee \bar{v}$	$t \vee \bar{w}$	$u \vee v \vee w \vee y$	$v \vee \bar{y}$
	$p, q$	$\bar{r}, \bar{s}$	$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$	$u, v$	$v, \bar{y}$
$\bar{p}$	$s, q$						
$\bar{q}$	$s, q$						
$s$		$\bar{r}, \bar{t}$					

Recall watched-literal invariant:  $\bar{x}_c, \bar{y}_c \notin I$  if possible.

## Watched-Literal Scheme: Example

$I$	Clauses and Watched Literals						
	$p \vee q \vee s$	$\bar{r} \vee \bar{s} \vee \bar{t}$	$t \vee \bar{u}$	$t \vee \bar{v}$	$t \vee \bar{w}$	$u \vee v \vee w \vee y$	$v \vee \bar{y}$
	$p, q$	$\bar{r}, \bar{s}$	$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$	$u, v$	$v, \bar{y}$
$\bar{p}$	$s, q$						
$\bar{q}$	$s, q$						
$s$		$\bar{r}, \bar{t}$					
$r$							

Recall watched-literal invariant:  $\bar{x}_c, \bar{y}_c \notin I$  if possible.

# Watched-Literal Scheme: Example

$I$	Clauses and Watched Literals						
	$p \vee q \vee s$	$\bar{r} \vee \bar{s} \vee \bar{t}$	$t \vee \bar{u}$	$t \vee \bar{v}$	$t \vee \bar{w}$	$u \vee v \vee w \vee y$	$v \vee \bar{y}$
	$p, q$	$\bar{r}, \bar{s}$	$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$	$u, v$	$v, \bar{y}$
$\bar{p}$	$s, q$						
$\bar{q}$	$s, q$						
$s$		$\bar{r}, \bar{t}$					
$r$		$\bar{r}, \bar{t}$					
$\bar{t}$							

Recall watched-literal invariant:  $\bar{x}_c, \bar{y}_c \notin I$  if possible.

# Watched-Literal Scheme: Example

$I$	Clauses and Watched Literals						
	$p \vee q \vee s$	$\bar{r} \vee \bar{s} \vee \bar{t}$	$t \vee \bar{u}$	$t \vee \bar{v}$	$t \vee \bar{w}$	$u \vee v \vee w \vee y$	$v \vee \bar{y}$
	$p, q$	$\bar{r}, \bar{s}$	$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$	$u, v$	$v, \bar{y}$
$\bar{p}$	$s, q$						
$\bar{q}$	$s, q$						
$s$		$\bar{r}, \bar{t}$					
$r$		$\bar{r}, \bar{t}$					
$\bar{t}$			$t, \bar{u}$				
$\bar{u}$							

Recall watched-literal invariant:  $\bar{x}_c, \bar{y}_c \notin I$  if possible.

# Watched-Literal Scheme: Example

$I$	Clauses and Watched Literals						
	$p \vee q \vee s$	$\bar{r} \vee \bar{s} \vee \bar{t}$	$t \vee \bar{u}$	$t \vee \bar{v}$	$t \vee \bar{w}$	$u \vee v \vee w \vee y$	$v \vee \bar{y}$
	$p, q$	$\bar{r}, \bar{s}$	$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$	$u, v$	$v, \bar{y}$
$\bar{p}$	$s, q$						
$\bar{q}$	$s, q$						
$s$		$\bar{r}, \bar{t}$					
$r$		$\bar{r}, \bar{t}$					
$\bar{t}$			$t, \bar{u}$	$t, \bar{v}$			
$\bar{u}$							
$\bar{v}$							

Recall watched-literal invariant:  $\bar{x}_c, \bar{y}_c \notin I$  if possible.



# Watched-Literal Scheme: Example

$I$	Clauses and Watched Literals						
	$p \vee q \vee s$	$\bar{r} \vee \bar{s} \vee \bar{t}$	$t \vee \bar{u}$	$t \vee \bar{v}$	$t \vee \bar{w}$	$u \vee v \vee w \vee y$	$v \vee \bar{y}$
	$p, q$	$\bar{r}, \bar{s}$	$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$	$u, v$	$v, \bar{y}$
$\bar{p}$	$s, q$						
$\bar{q}$	$s, q$						
$s$		$\bar{r}, \bar{t}$					
$r$		$\bar{r}, \bar{t}$					
$\bar{t}$			$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$		
$\bar{u}$							
$\bar{v}$							
$\bar{w}$							

Recall watched-literal invariant:  $\bar{x}_c, \bar{y}_c \notin I$  if possible.

# Watched-Literal Scheme: Example

$I$	Clauses and Watched Literals						
	$p \vee q \vee s$	$\bar{r} \vee \bar{s} \vee \bar{t}$	$t \vee \bar{u}$	$t \vee \bar{v}$	$t \vee \bar{w}$	$u \vee v \vee w \vee y$	$v \vee \bar{y}$
	$p, q$	$\bar{r}, \bar{s}$	$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$	$u, v$	$v, \bar{y}$
$\bar{p}$	$s, q$						
$\bar{q}$	$s, q$						
$s$		$\bar{r}, \bar{t}$					
$r$		$\bar{r}, \bar{t}$					
$\bar{t}$			$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$		
$\bar{u}$						$y, v$	
$\bar{v}$							
$\bar{w}$							

Recall watched-literal invariant:  $\bar{x}_c, \bar{y}_c \notin I$  if possible.

# Watched-Literal Scheme: Example

$I$	Clauses and Watched Literals						
	$p \vee q \vee s$	$\bar{r} \vee \bar{s} \vee \bar{t}$	$t \vee \bar{u}$	$t \vee \bar{v}$	$t \vee \bar{w}$	$u \vee v \vee w \vee y$	$v \vee \bar{y}$
	$p, q$	$\bar{r}, \bar{s}$	$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$	$u, v$	$v, \bar{y}$
$\bar{p}$	$s, q$						
$\bar{q}$	$s, q$						
$s$		$\bar{r}, \bar{t}$					
$r$		$\bar{r}, \bar{t}$					
$\bar{t}$			$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$		
$\bar{u}$						$y, v$	
$\bar{v}$						$y, v$	
$\bar{w}$							
$y$							

Recall watched-literal invariant:  $\bar{x}_c, \bar{y}_c \notin I$  if possible.

# Watched-Literal Scheme: Example

$I$	Clauses and Watched Literals						
	$p \vee q \vee s$	$\bar{r} \vee \bar{s} \vee \bar{t}$	$t \vee \bar{u}$	$t \vee \bar{v}$	$t \vee \bar{w}$	$u \vee v \vee w \vee y$	$v \vee \bar{y}$
	$p, q$	$\bar{r}, \bar{s}$	$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$	$u, v$	$v, \bar{y}$
$\bar{p}$	$s, q$						
$\bar{q}$	$s, q$						
$s$		$\bar{r}, \bar{t}$					
$r$		$\bar{r}, \bar{t}$					
$\bar{t}$			$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$		
$\bar{u}$						$y, v$	
$\bar{v}$						$y, v$	$v, \bar{y}$ ⚡
$\bar{w}$							
$y$							

Recall watched-literal invariant:  $\bar{x}_c, \bar{y}_c \notin I$  if possible.

# Watched-Literal Scheme: Example

$I$	Clauses and Watched Literals						
	$p \vee q \vee s$	$\bar{r} \vee \bar{s} \vee \bar{t}$	$t \vee \bar{u}$	$t \vee \bar{v}$	$t \vee \bar{w}$	$u \vee v \vee w \vee y$	$v \vee \bar{y}$
	$p, q$	$\bar{r}, \bar{s}$	$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$	$u, v$	$v, \bar{y}$
$\bar{p}$	$s, q$						
$\bar{q}$	$s, q$						
$s$		$\bar{r}, \bar{t}$					
$r$		$\bar{r}, \bar{t}$					
$\bar{t}$			$t, \bar{u}$	$t, \bar{v}$	$t, \bar{w}$		
$\bar{u}$						$y, v$	
$\bar{v}$						$y, v$	$v, \bar{y}$ ⚡
$\bar{w}$							
$y$							

Recall watched-literal invariant:  $\bar{x}_c, \bar{y}_c \notin I$  if possible.

Backtracking preserves this invariant!

# Outline

Satisfiability and Complexity Theory

Solver Ingredient 1: Watched-Literal Scheme

Solver Ingredient 2: Conflict-Driven Clause Learning

Other Solver Ingredients

Conclusion

Beyond SAT

# Conflict-Driven Clause Learning: Motivation

- Algorithm 2 with Watched-Literal Scheme still spends almost all its time on unit propagation.
- Suppose  $I = \{x_1, \dots, x_k\}$  leads to a conflict.  
That is:  $I$  falsifies some  $c \in \phi$ .
- Let's **learn from the conflict** to avoid similar mistakes later:
  - ▶ Find a **cause**  $\{x_{i_1}, \dots, x_{i_l}\} \subseteq I$  of the conflict.
  - ▶ Add **learnt clause**  $(\bar{x}_{i_1} \vee \dots \vee \bar{x}_{i_l})$  to avoid the conflict next time!
  - ▶ This avoids assignments  $x_{i_1}, \dots, x_{i_l}$  in the remaining search.
  - ▶ Note: must have  $\phi \models (\bar{x}_{i_1} \vee \dots \vee \bar{x}_{i_l})$ .

# SAT Algorithm 3: Adding Clause Learning

Let  $I = \{\}$ . Repeat:

1. Close  $I$  under unit propagation relative to  $\phi$
2. If conflict:
  - 2.1 Analyse conflict find the cause
  - 2.2 Backtrack to the appropriate level undo last decisions
  - 2.3 If there is none: return NO
  - 2.4 Add conflict clause to  $\phi$  new decision implicitly
3. Else:
  - 3.1 Select a variable  $x$  such that  $x, \bar{x} \notin I$
  - 3.2 If there is none: return YES
  - 3.3 Let  $I = I \cup \{\neg x\}$  make a decision



# Implication Graph

Maintain **implication graph** during unit propagation:

- For every decision  $x$ , create a node  $x$ .
- When  $(x_1 \vee \dots \vee x_{k+1})$  and  $\bar{x}_1, \dots, \bar{x}_k$  produce  $x_{k+1}$ :
  - ▶ Add a node  $x_{k+1}$ .
  - ▶ Add edges  $(\bar{x}_i, x_{k+1})$  and add  $c$  to their label set.
- When  $(x_1 \vee \dots \vee x_k)$  and  $\bar{x}_1, \dots, \bar{x}_k$  produce a conflict:
  - ▶ Add a node  $x_k$ .
  - ▶ Add edges  $(\bar{x}_i, x_k)$ .
  - ▶ Add edges  $(x_k, \perp)$  and  $(\bar{x}_k, \perp)$ .

# Implication Graph

Maintain **implication graph** during unit propagation:

- For every decision  $x$ , create a node  $x$ .
- When  $(x_1 \vee \dots \vee x_{k+1})$  and  $\bar{x}_1, \dots, \bar{x}_k$  produce  $x_{k+1}$ :
  - ▶ Add a node  $x_{k+1}$ .
  - ▶ Add edges  $(\bar{x}_i, x_{k+1})$  and add  $c$  to their label set.
- When  $(x_1 \vee \dots \vee x_k)$  and  $\bar{x}_1, \dots, \bar{x}_k$  produce a conflict:
  - ▶ Add a node  $x_k$ .
  - ▶ Add edges  $(\bar{x}_i, x_k)$ .
  - ▶ Add edges  $(x_k, \perp)$  and  $(\bar{x}_k, \perp)$ .

Find **conflict clause**:

- Let  $\{(x_1, y_1), \dots, (x_k, y_k)\}$  be a cut separating decisions from  $\perp$ .
- Then  $(x_1 \wedge \dots \wedge x_k)$  is a cause of the conflict.
- Conflict clause:  $(\bar{x}_1 \vee \dots \vee \bar{x}_k)$ .

# Implication Graph

Maintain **implication graph** during unit propagation:

- For every decision  $x$ , create a node  $x$ .
- When  $(x_1 \vee \dots \vee x_{k+1})$  and  $\bar{x}_1, \dots, \bar{x}_k$  produce  $x_{k+1}$ :
  - ▶ Add a node  $x_{k+1}$ .
  - ▶ Add edges  $(\bar{x}_i, x_{k+1})$  and add  $c$  to their label set.
- When  $(x_1 \vee \dots \vee x_k)$  and  $\bar{x}_1, \dots, \bar{x}_k$  produce a conflict:
  - ▶ Add a node  $x_k$ .
  - ▶ Add edges  $(\bar{x}_i, x_k)$ .
  - ▶ Add edges  $(x_k, \perp)$  and  $(\bar{x}_k, \perp)$ .

Find **conflict clause**:

- Let  $\{(x_1, y_1), \dots, (x_k, y_k)\}$  be a cut separating decisions from  $\perp$ .
- Then  $(x_1 \wedge \dots \wedge x_k)$  is a cause of the conflict.
- Conflict clause:  $(\bar{x}_1 \vee \dots \vee \bar{x}_k)$ .

Good scheme is **FirstUIP**: Cut such that  $x_i$  are:

- The UIP: Node through which all paths from last decision to  $\perp$  go.
- Literals from earlier levels if necessary.

# Outline

Satisfiability and Complexity Theory

Solver Ingredient 1: Watched-Literal Scheme

Solver Ingredient 2: Conflict-Driven Clause Learning

**Other Solver Ingredients**

Conclusion

Beyond SAT

# Variable Selection Heuristic

Recall algorithm:

3.1 Select a variable  $x$  such that  $x, \bar{x} \notin I$

3.3 Let  $I = I \cup \{\neg x\}$

■ Variable Selection Heuristics rank variables by attractiveness:

- ▶ Maximum Occurrence in clauses of Minimum size (MOM):
  - Prefer literals that occur often in small clauses.
- ▶ Dynamic Largest Individual Sum (DLIS):
  - Select variable that occurs most frequently in unsatisfied clauses.
- ▶ Variable State Independent Decaying Sum (VSIDS):
  - Score variables numerically.
  - Each time  $x$  occurs in conflict, increase its score.
  - Decay scores to increase meaning of recent conflicts.

# Direction Heuristics

Recall algorithm:

3.1 Select a variable  $x$  such that  $x, \bar{x} \notin I$

3.3 Let  $I = I \cup \{\neg x\}$

■ Branching to  $\neg x$  is better than branching to  $x$ :

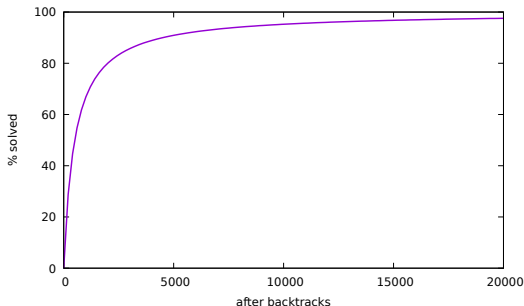
- ▶ Keeps the search in the same search space.
- ▶ Most things in the real world are false.

■ Phase-Saving Heuristic:

- ▶ Remember the last value assigned to  $x$ .
- ▶ Assign  $x$  this value in 3.3.

# Randomized Restarts

## ■ Heavy-tail behaviour typical:



## ■ Restarts avoid getting stuck in long tail:

- ▶ Restart after  $N$  conflicts.
- ▶ Keep the learnt clauses.
- ▶ Dynamically grow  $N$  exponentially – otherwise incomplete.

## ■ $N$ differs a lot between solvers:

- ▶ Initial  $N$  may vary from high hundreds to low tens.
- ▶ Trend is to smaller  $N$ .

# Outline

Satisfiability and Complexity Theory

Solver Ingredient 1: Watched-Literal Scheme

Solver Ingredient 2: Conflict-Driven Clause Learning

Other Solver Ingredients

Conclusion

Beyond SAT



# Enumerating Models

Suppose the SAT solver computes a satisfying assignment  $I$ .

How to find another one?

# Satisfiability for $\leq$ 2-Clauses

## Theorem: complexity of 2-SAT

2-SAT can be solved in polynomial time.

- Input: 2-CNF formula  $\phi$  of length  $n$ .
- Let  $\phi^*$  be the least set such that
  - ▶  $\phi \subseteq \phi^*$  and
  - ▶ if  $(x \vee y), (\bar{x} \vee z) \in \phi^*$ , then  $(y \vee z) \in \phi^*$ .
- $\phi$  is equivalent to  $\phi^*$ .
- $\phi$  is unsatisfiable iff  $(x \vee x), (\neg x \vee \neg x) \in \phi^*$  for some  $x$ .
- $|\{x \mid x \text{ literal in } \phi\}| \leq n$ .
- $|\{(x \vee y, \bar{x} \vee z) \mid y, z \text{ literal in } \phi\}| \leq n^2$  for each literal  $x$ .

# Benchmarks

Benchmark	#Variables	#Clauses	Naive	WSL	CDCL
aim-50-1_6-yes1-4	50	80	—	$3 \times 10^{-3}$	$3 \times 10^{-4}$
uf75-013	75	325	—	$4 \times 10^{-4}$	$2 \times 10^{-4}$
pigeon/12	156	949	—	—	2.0
example2_gr_rcs_w6	2,664	27,684	—	—	$3 \times 10^{-3}$
fvp-unsat	24,065	731,850	—	—	—
cache_10	227,210	879,754	—	—	86.4

Execution time (in seconds) of different algorithms:

- Naive: Slide 21
- WSL: Slide 25 with watched literal scheme
- CDCL: Slide 31 with watched literal scheme, FirstUIP clause learning

Timeout after 600 seconds.

Most of the files are from othe 2002 SAT competition.

# Pigeonhole Principle

**Pigeonhole principle:**  $k + 1$  pigeons cannot be put in  $k$  holes.

$$k = 1 : \quad (h_1 = p_1) \wedge \\ (h_1 = p_2)$$

where we assume  $p_i \neq p_j$  for  $i \neq j$ .

# Pigeonhole Principle

**Pigeonhole principle:**  $k + 1$  pigeons cannot be put in  $k$  holes.

$$\begin{aligned}k = 2 : \quad & (h_1 = p_1 \vee h_2 = p_1) \wedge \\ & (h_1 = p_2 \vee h_2 = p_2) \wedge \\ & (h_1 = p_3 \vee h_2 = p_3)\end{aligned}$$

where we assume  $p_i \neq p_j$  for  $i \neq j$ .

# Pigeonhole Principle

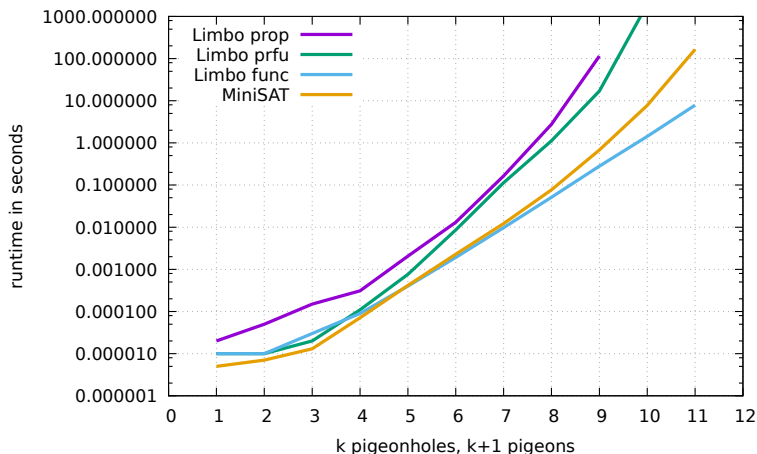
**Pigeonhole principle:**  $k + 1$  pigeons cannot be put in  $k$  holes.

$$\begin{aligned}k = 3 : \quad & (h_1 = p_1 \vee h_2 = p_1 \vee h_3 = p_1) \wedge \\& (h_1 = p_2 \vee h_2 = p_2 \vee h_3 = p_2) \wedge \\& (h_1 = p_3 \vee h_2 = p_3 \vee h_3 = p_3) \wedge \\& (h_1 = p_4 \vee h_2 = p_4 \vee h_3 = p_4)\end{aligned}$$

where we assume  $p_i \neq p_j$  for  $i \neq j$ .

# Pigeonhole Principle

**Pigeonhole principle:**  $k + 1$  pigeons cannot be put in  $k$  holes.



# Summary SAT

- SAT is provably hard:
  - ▶ SAT is **NP-complete** – believed to take exponential time.
  - ▶ All problems in **NP can be reduced to SAT** efficiently.
  - ▶ SAT is perhaps the best-understood problem of complexity theory.



# Summary SAT

- SAT is provably hard:
  - ▶ SAT is **NP-complete** – believed to take exponential time.
  - ▶ All problems in **NP can be reduced to SAT** efficiently.
  - ▶ SAT is perhaps the best-understood problem of complexity theory.
- SAT is often feasible in practice:
  - ▶ SAT solvers are far-developed.
  - ▶ Hence SAT is an attractive target for reductions.
  - ▶ Real-world instances often have **easy structure**.
  - ▶ There are small but very hard instances though.

# Summary SAT

- SAT is provably hard:
  - ▶ SAT is **NP-complete** – believed to take exponential time.
  - ▶ All problems in **NP can be reduced to SAT** efficiently.
  - ▶ SAT is perhaps the best-understood problem of complexity theory.
- SAT is often feasible in practice:
  - ▶ SAT solvers are far-developed.
  - ▶ Hence SAT is an attractive target for reductions.
  - ▶ Real-world instances often have **easy structure**.
  - ▶ There are small but very hard instances though.
- Key ingredients for a fast SAT solver:
  - ▶ **Watched-Literal Scheme**
  - ▶ **Clause Learning**
  - ▶ **Variable Selection Heuristic**
  - ▶ **Random Restarts**

# Outline

Satisfiability and Complexity Theory

Solver Ingredient 1: Watched-Literal Scheme

Solver Ingredient 2: Conflict-Driven Clause Learning

Other Solver Ingredients

Conclusion

Beyond SAT

# Quantified Boolean Formulas

Syntax: propositional logic  $\exists x \phi$   $\forall x \phi$

# Quantified Boolean Formulas

Syntax: propositional logic  $\exists x \phi$   $\forall x \phi$

Semantics:

- $x$  is true iff  $x = \text{TRUE}$
- $\neg \phi$  is true iff  $\phi$  is not true
- $(\phi_1 \vee \phi_2)$  is true iff  $\phi_1$  or  $\phi_2$  is true
- $(\phi_1 \wedge \phi_2)$  is true iff  $\phi_1$  and  $\phi_2$  are true
- $\exists x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  or  $\phi_{\text{TRUE}}^x$  is true
- $\forall x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  and  $\phi_{\text{TRUE}}^x$  are true

where  $\phi_y^x$  is  $\phi$  with  $x$  replaced by  $y$ .

# Quantified Boolean Formulas

Syntax: propositional logic  $\exists x \phi$   $\forall x \phi$

Semantics:

- $x$  is true iff  $x = \text{TRUE}$
- $\neg \phi$  is true iff  $\phi$  is not true
- $(\phi_1 \vee \phi_2)$  is true iff  $\phi_1$  or  $\phi_2$  is true
- $(\phi_1 \wedge \phi_2)$  is true iff  $\phi_1$  and  $\phi_2$  are true
- $\exists x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  or  $\phi_{\text{TRUE}}^x$  is true
- $\forall x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  and  $\phi_{\text{TRUE}}^x$  are true

where  $\phi_y^x$  is  $\phi$  with  $x$  replaced by  $y$ .

Ex.: We abbreviate FALSE, TRUE by F, T:

$\forall x \exists y ((\neg x \vee y) \wedge (x \vee \neg y))$  is true

# Quantified Boolean Formulas

Syntax: propositional logic  $\exists x \phi$   $\forall x \phi$

Semantics:

- $x$  is true iff  $x = \text{TRUE}$
- $\neg \phi$  is true iff  $\phi$  is not true
- $(\phi_1 \vee \phi_2)$  is true iff  $\phi_1$  or  $\phi_2$  is true
- $(\phi_1 \wedge \phi_2)$  is true iff  $\phi_1$  and  $\phi_2$  are true
- $\exists x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  or  $\phi_{\text{TRUE}}^x$  is true
- $\forall x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  and  $\phi_{\text{TRUE}}^x$  are true

where  $\phi_y^x$  is  $\phi$  with  $x$  replaced by  $y$ .

Ex.: We abbreviate FALSE, TRUE by F, T:

$\forall x \exists y ((\neg x \vee y) \wedge (x \vee \neg y))$  is true

iff  $\exists y ((\neg F \vee y) \wedge (F \vee \neg y))$  is true and  
 $\exists y ((\neg T \vee y) \wedge (T \vee \neg y))$  is true

# Quantified Boolean Formulas

Syntax: propositional logic  $\exists x \phi$   $\forall x \phi$

Semantics:

- $x$  is true iff  $x = \text{TRUE}$
- $\neg \phi$  is true iff  $\phi$  is not true
- $(\phi_1 \vee \phi_2)$  is true iff  $\phi_1$  or  $\phi_2$  is true
- $(\phi_1 \wedge \phi_2)$  is true iff  $\phi_1$  and  $\phi_2$  are true
- $\exists x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  or  $\phi_{\text{TRUE}}^x$  is true
- $\forall x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  and  $\phi_{\text{TRUE}}^x$  are true

where  $\phi_y^x$  is  $\phi$  with  $x$  replaced by  $y$ .

Ex.: We abbreviate FALSE, TRUE by F, T:

$\forall x \exists y ((\neg x \vee y) \wedge (x \vee \neg y))$  is true

iff  $\exists y ((\neg F \vee y) \wedge (F \vee \neg y))$  is true and  
 $\exists y ((\neg T \vee y) \wedge (T \vee \neg y))$  is true

iff  $((\neg F \vee F) \wedge (F \vee \neg F))$  or  $((\neg F \vee T) \wedge (F \vee \neg T))$  is true) and  
 $((\neg T \vee F) \wedge (T \vee \neg F))$  or  $((\neg T \vee T) \wedge (T \vee \neg T))$  is true)



# Quantified Boolean Formulas

Syntax: propositional logic  $\exists x \phi$   $\forall x \phi$

Semantics:

- $x$  is true iff  $x = \text{TRUE}$
- $\neg \phi$  is true iff  $\phi$  is not true
- $(\phi_1 \vee \phi_2)$  is true iff  $\phi_1$  or  $\phi_2$  is true
- $(\phi_1 \wedge \phi_2)$  is true iff  $\phi_1$  and  $\phi_2$  are true
- $\exists x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  or  $\phi_{\text{TRUE}}^x$  is true
- $\forall x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  and  $\phi_{\text{TRUE}}^x$  are true

where  $\phi_y^x$  is  $\phi$  with  $x$  replaced by  $y$ .

Ex.: We abbreviate FALSE, TRUE by F, T:

$\forall x \exists y ((\neg x \vee y) \wedge (x \vee \neg y))$  is true

iff  $\exists y ((\neg F \vee y) \wedge (F \vee \neg y))$  is true and  
 $\exists y ((\neg T \vee y) \wedge (T \vee \neg y))$  is true

iff  $((\neg F \vee F) \wedge (F \vee \neg F))$  or  $((\neg F \vee T) \wedge (F \vee \neg T))$  is true) and  
 $((\neg T \vee F) \wedge (T \vee \neg F))$  or  $((\neg T \vee T) \wedge (T \vee \neg T))$  is true)

# Quantified Boolean Formulas

Syntax: propositional logic  $\exists x \phi$   $\forall x \phi$

Semantics:

- $x$  is true iff  $x = \text{TRUE}$
- $\neg \phi$  is true iff  $\phi$  is not true
- $(\phi_1 \vee \phi_2)$  is true iff  $\phi_1$  or  $\phi_2$  is true
- $(\phi_1 \wedge \phi_2)$  is true iff  $\phi_1$  and  $\phi_2$  are true
- $\exists x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  or  $\phi_{\text{TRUE}}^x$  is true
- $\forall x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  and  $\phi_{\text{TRUE}}^x$  are true

where  $\phi_y^x$  is  $\phi$  with  $x$  replaced by  $y$ .

Ex.: We abbreviate FALSE, TRUE by F, T:

$\forall x \exists y ((\neg x \vee y) \wedge (x \vee \neg y))$  is true

iff  $\exists y ((\neg F \vee y) \wedge (F \vee \neg y))$  is true and  
 $\exists y ((\neg T \vee y) \wedge (T \vee \neg y))$  is true

iff  $((\neg F \vee F) \wedge (F \vee \neg F))$  or  $((\neg F \vee T) \wedge (F \vee \neg T))$  is true and  
 $((\neg T \vee F) \wedge (T \vee \neg F))$  or  $((\neg T \vee T) \wedge (T \vee \neg T))$  is true

# Quantified Boolean Formulas

Syntax: propositional logic  $\exists x \phi$   $\forall x \phi$

Semantics:

- $x$  is true iff  $x = \text{TRUE}$
- $\neg \phi$  is true iff  $\phi$  is not true
- $(\phi_1 \vee \phi_2)$  is true iff  $\phi_1$  or  $\phi_2$  is true
- $(\phi_1 \wedge \phi_2)$  is true iff  $\phi_1$  and  $\phi_2$  are true
- $\exists x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  or  $\phi_{\text{TRUE}}^x$  is true
- $\forall x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  and  $\phi_{\text{TRUE}}^x$  are true

where  $\phi_y^x$  is  $\phi$  with  $x$  replaced by  $y$ .

Ex.: We abbreviate FALSE, TRUE by F, T:

$\forall x \exists y ((\neg x \vee y) \wedge (x \vee \neg y))$  is true

iff  $\exists y ((\neg F \vee y) \wedge (F \vee \neg y))$  is true and  
 $\exists y ((\neg T \vee y) \wedge (T \vee \neg y))$  is true

iff  $((\neg F \vee F) \wedge (F \vee \neg F))$  or  $((\neg F \vee T) \wedge (F \vee \neg T))$  is true and  
 $((\neg T \vee F) \wedge (T \vee \neg F))$  or  $((\neg T \vee T) \wedge (T \vee \neg T))$  is true

# Quantified Boolean Formulas

Syntax: propositional logic  $\exists x \phi$   $\forall x \phi$

Semantics:

- $x$  is true iff  $x = \text{TRUE}$
- $\neg \phi$  is true iff  $\phi$  is not true
- $(\phi_1 \vee \phi_2)$  is true iff  $\phi_1$  or  $\phi_2$  is true
- $(\phi_1 \wedge \phi_2)$  is true iff  $\phi_1$  and  $\phi_2$  are true
- $\exists x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  or  $\phi_{\text{TRUE}}^x$  is true
- $\forall x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  and  $\phi_{\text{TRUE}}^x$  are true

where  $\phi_y^x$  is  $\phi$  with  $x$  replaced by  $y$ .

Note: For propositional  $\phi$ :

- $\phi$  is satisfiable iff  $\exists x_1 \dots \exists x_k \phi$  is true
- $\phi$  is valid iff  $\forall x_1 \dots \forall x_k \phi$  is true

# Quantified Boolean Formulas

Syntax: propositional logic  $\exists x \phi$   $\forall x \phi$

Semantics:

- $x$  is true iff  $x = \text{TRUE}$
- $\neg \phi$  is true iff  $\phi$  is not true
- $(\phi_1 \vee \phi_2)$  is true iff  $\phi_1$  or  $\phi_2$  is true
- $(\phi_1 \wedge \phi_2)$  is true iff  $\phi_1$  and  $\phi_2$  are true
- $\exists x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  or  $\phi_{\text{TRUE}}^x$  is true
- $\forall x \phi$  is true iff  $\phi_{\text{FALSE}}^x$  and  $\phi_{\text{TRUE}}^x$  are true

where  $\phi_y^x$  is  $\phi$  with  $x$  replaced by  $y$ .

Note: For propositional  $\phi$ :

- $\phi$  is satisfiable iff  $\exists x_1 \dots \exists x_k \phi$  is true
- $\phi$  is valid iff  $\forall x_1 \dots \forall x_k \phi$  is true

**Theorem: complexity of QBF**

Deciding whether a QBF is true is PSPACE-complete.

# Model Counting

How many satisfying assignment does a formula have?

- Exact solvers struggle with huge solution space.
- Approximative solvers sample solution space.
- Useful for probabilistic reasoning.

# Satisfiability Modulo Theories

Combine SAT solver with a theory solver (e.g., linear equations).

- $(x + y \geq 3) \vee (x + y \leq 1)$
- $(x < 1) \rightarrow (y < 3)$

# Satisfiability Modulo Theories

Combine SAT solver with a theory solver (e.g., linear equations).

- $(x + y \geq 3) \vee (x + y \leq 1)$
- $(x < 1) \rightarrow (y < 3)$
- $(x < 1) \wedge (y < 3) \rightarrow (x + y < 3)$



# Satisfiability Modulo Theories

Combine SAT solver with a theory solver (e.g., linear equations).

- $(x + y \geq 3) \vee (-x - y \geq -1)$
- $(x \geq 1) \vee \neg(y \geq 3)$
- $(x \geq 1) \vee (y \geq 3) \vee \neg(x + y \geq 3)$

# Satisfiability Modulo Theories

Combine SAT solver with a theory solver (e.g., linear equations).

■  $(x + y \geq 3) \vee (-x - y \geq -1)$

■  $(x \geq 1) \vee \neg(y \geq 3)$

■  $(x \geq 1) \vee (y \geq 3) \vee \neg(x + y \geq 3)$

■  $p \vee q$

■  $r \vee \neg s$

■  $r \vee s \vee \neg p$

■  $\bar{r}, \bar{s}, \bar{p}, q \Rightarrow (x < 1), (y < 3), (x + y < 3)$  ✗

■  $r, s, p, q \Rightarrow (x \geq 1), (y \geq 3), (x + y \geq 3), (x + y \leq 1)$  ✗

■  $r, s, \bar{p}, q \Rightarrow (x \geq 1), (y \geq 3), (x + y < 3), (x + y \leq 1)$  ✓