

# Generalized SAT Solving in a First-Order Logic Framework

Christoph Zengler

University of Passau

12.12.2007

# Propositional Logic

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic

The SAT Problem

Overview

## Efficient SAT Solving

Unit Propagation

Branching Heuristics

Conflict Driven

Clause Learning

Benchmarks

## The next step: Q-SAT

The Q-SAT Problem

An Algorithm for Q-SAT

Benchmarks

## Parametric Q-SAT

Motivation

The Algorithm

## Conclusion

- a set of Variables  $\mathcal{V}$  (literals)
- constants  $\top$  and  $\perp$
- boolean operations  $\neg, \wedge, \vee, \implies$  and  $\iff$

## Syntax

Formulas are defined inductively:

- $\top$  and  $\perp$  are formulas

For formulas  $\varphi$  and  $\psi$  also

- $\neg\varphi$
- $(\varphi \wedge \psi)$
- $(\varphi \vee \psi)$
- $(\varphi \implies \psi)$
- $(\varphi \iff \psi)$

are formulas.

# Propositional Logic

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic

The SAT Problem

Overview

## Efficient SAT Solving

Unit Propagation

Branching Heuristics

Conflict Driven

Clause Learning

Benchmarks

## The next step: Q-SAT

The Q-SAT Problem

An Algorithm for Q-SAT

Benchmarks

## Parametric Q-SAT

Motivation

The Algorithm

## Conclusion

- a set of Variables  $\mathcal{V}$  (literals)
- constants  $\top$  and  $\perp$
- boolean operations  $\neg, \wedge, \vee, \implies$  and  $\iff$

## Syntax

Formulas are defined inductively:

- $\top$  and  $\perp$  are formulas

For formulas  $\varphi$  and  $\psi$  also

- $\neg\varphi$
- $(\varphi \wedge \psi)$
- $(\varphi \vee \psi)$
- $(\varphi \implies \psi)$
- $(\varphi \iff \psi)$

are formulas.

# Propositional Logic

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic

The SAT Problem

Overview

Efficient SAT  
Solving

Unit Propagation

Branching Heuristics

Conflict Driven

Clause Learning

Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem

An Algorithm for  
Q-SAT

Benchmarks

Parametric  
Q-SAT

Motivation

The Algorithm

Conclusion

## Interpretation of a formula

possibly partial assignment  $\alpha$ , mapping  $x \in \mathcal{V}$  to a truth value

$$\alpha(x) \in \{\top, \perp\}$$

Notation:  $[x_1 \leftarrow \alpha(x_1), x_2 \leftarrow \alpha(x_2), \dots, x_k \leftarrow \alpha(x_k)]$

- Simplification of formulas by laws of Boolean Algebra
- Deriving of a truth value for a formula
- $\alpha \models \varphi$ :  $\varphi$  is true under assignment  $\alpha$

## Satisfiability

- $\varphi$  is *satisfiable*: there exists an interpretation  $\alpha \models \varphi$
- $\varphi$  is *unsatisfiable*: there exists no interpretation  $\alpha \models \varphi$

# Propositional Logic

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic

The SAT Problem

Overview

Efficient SAT  
Solving

Unit Propagation

Branching Heuristics

Conflict Driven

Clause Learning

Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem

An Algorithm for  
Q-SAT

Benchmarks

Parametric  
Q-SAT

Motivation

The Algorithm

Conclusion

## Interpretation of a formula

possibly partial assignment  $\alpha$ , mapping  $x \in \mathcal{V}$  to a truth value

$$\alpha(x) \in \{\top, \perp\}$$

Notation:  $[x_1 \leftarrow \alpha(x_1), x_2 \leftarrow \alpha(x_2), \dots, x_k \leftarrow \alpha(x_k)]$

- Simplification of formulas by laws of Boolean Algebra
- Deriving of a truth value for a formula
- $\alpha \models \varphi$ :  $\varphi$  is true under assignment  $\alpha$

## Satisfiability

- $\varphi$  is *satisfiable*: there exists an interpretation  $\alpha \models \varphi$
- $\varphi$  is *unsatisfiable*: there exists no interpretation  $\alpha \models \varphi$

# Propositional Logic

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic

The SAT Problem

Overview

Efficient SAT  
Solving

Unit Propagation

Branching Heuristics

Conflict Driven

Clause Learning

Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem

An Algorithm for  
Q-SAT

Benchmarks

Parametric  
Q-SAT

Motivation

The Algorithm

Conclusion

## Interpretation of a formula

possibly partial assignment  $\alpha$ , mapping  $x \in \mathcal{V}$  to a truth value

$$\alpha(x) \in \{\top, \perp\}$$

Notation:  $[x_1 \leftarrow \alpha(x_1), x_2 \leftarrow \alpha(x_2), \dots, x_k \leftarrow \alpha(x_k)]$

- Simplification of formulas by laws of Boolean Algebra
- Deriving of a truth value for a formula
- $\alpha \models \varphi$ :  $\varphi$  is true under assignment  $\alpha$

## Satisfiability

- $\varphi$  is *satisfiable*: there exists an interpretation  $\alpha \models \varphi$
- $\varphi$  is *unsatisfiable*: there exists no interpretation  $\alpha \models \varphi$

# Examples

## Generalized SAT Solving

Christoph Zengler

## Introduction

### Propositional Logic

The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

### Example (Formula in Propositional Logic)

$$\varphi := (a \wedge b) \vee ((a \implies \neg c) \wedge (b \iff c))$$

- $\mathcal{V}(\varphi) = \{a, b, c\}$

### Example (Interpretation)

- $\alpha := [a \leftarrow \perp, b \leftarrow \top, c \leftarrow \top]$

$$\begin{aligned}\varphi &:= (\perp \wedge \top) \vee ((\perp \implies \neg \top) \wedge (\top \iff \top)) = \top \\ &= \perp \vee (\top \wedge \top) \\ &= \perp \vee \top \\ &= \top\end{aligned}$$

# Examples

## Generalized SAT Solving

Christoph Zengler

## Introduction

### Propositional Logic

The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

### Example (Formula in Propositional Logic)

$$\varphi := (a \wedge b) \vee ((a \implies \neg c) \wedge (b \iff c))$$

- $\mathcal{V}(\varphi) = \{a, b, c\}$

### Example (Interpretation)

- $\alpha := [a \leftarrow \perp, b \leftarrow \top, c \leftarrow \top]$

$$\begin{aligned}\varphi &:= (\perp \wedge \top) \vee ((\perp \implies \neg \top) \wedge (\top \iff \top)) = \top \\ &= \perp \vee (\top \wedge \top) \\ &= \perp \vee \top \\ &= \top\end{aligned}$$

# CNF and Satisfiability

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic

The SAT Problem

Overview

Efficient SAT  
Solving

Unit Propagation

Branching Heuristics

Conflict Driven

Clause Learning

Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem

An Algorithm for  
Q-SAT

Benchmarks

Parametric  
Q-SAT

Motivation

The Algorithm

Conclusion

## CNF

**CNF:** Conjunction of clauses. **Clause:** Disjunction of literals  $x_{ij}$

$$\varphi := \bigwedge_{i=0}^n \bigvee_{j=0}^{m_i} x_{ij}$$

## Example (Satisfiability)

$$(a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee \neg b)$$

- Satisfiable:  $[a \leftarrow \top, b \leftarrow \perp]$

$$(a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b)$$

- Unsatisfiable: No satisfying assignment

# CNF and Satisfiability

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic

The SAT Problem

Overview

Efficient SAT  
Solving

Unit Propagation

Branching Heuristics

Conflict Driven

Clause Learning

Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem

An Algorithm for  
Q-SAT

Benchmarks

Parametric  
Q-SAT

Motivation

The Algorithm

Conclusion

## CNF

**CNF:** Conjunction of clauses. **Clause:** Disjunction of literals  $x_{ij}$

$$\varphi := \bigwedge_{i=0}^n \bigvee_{j=0}^{m_i} x_{ij}$$

## Example (Satisfiability)

$$(a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee \neg b)$$

- Satisfiable:  $[a \leftarrow \top, b \leftarrow \perp]$

$$(a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee \neg b)$$

- Unsatisfiable: No satisfying assignment

# CNF and Satisfiability

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic

The SAT Problem

Overview

Efficient SAT  
Solving

Unit Propagation

Branching Heuristics

Conflict Driven

Clause Learning

Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem

An Algorithm for  
Q-SAT

Benchmarks

Parametric  
Q-SAT

Motivation

The Algorithm

Conclusion

## CNF

**CNF:** Conjunction of clauses. **Clause:** Disjunction of literals  $x_{ij}$

$$\varphi := \bigwedge_{i=0}^n \bigvee_{j=0}^{m_i} x_{ij}$$

## Example (Satisfiability)

$$(a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee \neg b)$$

- Satisfiable:  $[a \leftarrow \top, b \leftarrow \perp]$

$$(a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b)$$

- Unsatisfiable: No satisfying assignment

# The SAT Problem

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic

The SAT Problem

Overview

## Efficient SAT Solving

Unit Propagation

Branching Heuristics

Conflict Driven

Clause Learning

Benchmarks

## The next step: Q-SAT

The Q-SAT Problem

An Algorithm for Q-SAT

Benchmarks

## Parametric Q-SAT

Motivation

The Algorithm

## Conclusion

## SAT

The Problem of the **SAT**isfiability of a formula is called the *SAT Problem*. A SAT Solver decides for a given input formula  $\varphi$  if there is a satisfying assignment or not.

## Theorem

*SAT is NP-complete*

- Was the first problem to be shown as NP-complete (Cook, 1971)
- Can be reduced to any other problem in NP in polynomial time
- Fast SAT checking algorithms induce fast algorithms for other NP problems

## $k$ -SAT

for  $k > 2$ ,  $k$ -SAT is also NP-complete.

( $k$ -SAT: the input formula is in CNF with a maximum of  $k$  literals per clause)

# The SAT Problem

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic

The SAT Problem

Overview

## Efficient SAT Solving

Unit Propagation

Branching Heuristics

Conflict Driven

Clause Learning

Benchmarks

## The next step: Q-SAT

The Q-SAT Problem

An Algorithm for  
Q-SAT

Benchmarks

## Parametric Q-SAT

Motivation

The Algorithm

## Conclusion

## SAT

The Problem of the **SAT**isfiability of a formula is called the *SAT Problem*. A SAT Solver decides for a given input formula  $\varphi$  if there is a satisfying assignment or not.

## Theorem

*SAT is NP-complete*

- Was the first problem to be shown as NP-complete (Cook, 1971)
- Can be reduced to any other problem in NP in polynomial time
- Fast SAT checking algorithms induce fast algorithms for other NP problems

## $k$ -SAT

for  $k > 2$ ,  $k$ -SAT is also NP-complete.

( $k$ -SAT: the input formula is in CNF with a maximum of  $k$  literals per clause)

# The SAT Problem

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic

The SAT Problem

Overview

## Efficient SAT Solving

Unit Propagation

Branching Heuristics

Conflict Driven

Clause Learning

Benchmarks

## The next step: Q-SAT

The Q-SAT Problem

An Algorithm for Q-SAT

Benchmarks

## Parametric Q-SAT

Motivation

The Algorithm

## Conclusion

## SAT

The Problem of the **SAT**isfiability of a formula is called the *SAT Problem*. A SAT Solver decides for a given input formula  $\varphi$  if there is a satisfying assignment or not.

## Theorem

*SAT is NP-complete*

- Was the first problem to be shown as NP-complete (Cook, 1971)
- Can be reduced to any other problem in NP in polynomial time
- Fast SAT checking algorithms induce fast algorithms for other NP problems

## *k*-SAT

for  $k > 2$ , *k*-SAT is also NP-complete.

(*k*-SAT: the input formula is in CNF with a maximum of  $k$  literals per clause)

# The SAT Problem

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic

The SAT Problem

Overview

## Efficient SAT Solving

Unit Propagation

Branching Heuristics

Conflict Driven

Clause Learning

Benchmarks

## The next step: Q-SAT

The Q-SAT Problem

An Algorithm for Q-SAT

Benchmarks

## Parametric Q-SAT

Motivation

The Algorithm

## Conclusion

### SAT

The Problem of the **SAT**isfiability of a formula is called the *SAT Problem*. A SAT Solver decides for a given input formula  $\varphi$  if there is a satisfying assignment or not.

### Theorem

*SAT is NP-complete*

- Was the first problem to be shown as NP-complete (Cook, 1971)
- Can be reduced to any other problem in NP in polynomial time
- Fast SAT checking algorithms induce fast algorithms for other NP problems

### $k$ -SAT

for  $k > 2$ ,  $k$ -SAT is also NP-complete.

( $k$ -SAT: the input formula in in CNF with a maximum of  $k$  literals per clause)

# Overview

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem

Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

Where is the link to First-Order Logic and to Quantifier Elimination?

# Overview

Where is the link to First-Order Logic and to Quantifier Elimination?

Formulas with only existential variables (no free variables)

SAT

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem

Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# Overview

Where is the link to First-Order Logic and to Quantifier Elimination?

Formulas with only existential variables (no free variables)

SAT

Formulas with existential and universal variables (no free variables)

Q-SAT

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem

Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# Overview

Where is the link to First-Order Logic and to Quantifier Elimination?

Formulas with only existential variables (no free variables)

SAT

Formulas with existential and universal variables (no free variables)

Q-SAT

Arbitrary quantified formulas

parametric Q-SAT

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem

Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# Overview

Where is the link to First-Order Logic and to Quantifier Elimination?

Formulas with only existential variables (no free variables)

SAT

Formulas with existential and universal variables (no free variables)

Q-SAT

Arbitrary quantified formulas  
parametric Q-SAT

QE

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem

Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# DLL Algorithm

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem

Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

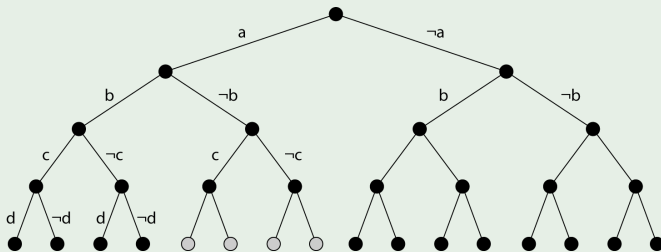
Motivation  
The Algorithm

Conclusion

- Problem can be solved naively in  $O(2^n)$  by testing all assignments

Example (Binary Search tree)

$$\varphi := (a \vee b \vee c) \wedge (a \vee \neg c) \wedge (a \vee \neg b) \wedge (\neg a \vee \neg b) \wedge (d \vee \neg d)$$



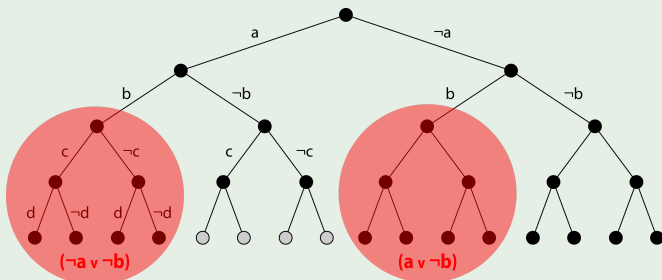
# DLL Algorithm

- Improvement: If an assignment already yields in  $\perp$  we can cut the search tree at this node

⇒ Backtracking with Branch & Bound

## Example (Binary Search tree)

$$\varphi := (a \vee b \vee c) \wedge (a \vee \neg c) \wedge (a \vee \neg b) \wedge (\neg a \vee \neg b) \wedge (d \vee \neg d)$$



Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem

Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# DLL Algorithm

## DLL Algorithm after Davis, Logeman, Loveland

### DLL

**Input:** Formula  $\varphi$  in Propositional Logic, interpretation  $\alpha$

```
DLL( $\varphi, \alpha$ ) if ( $\varphi, \alpha$ ) =  $\perp$  then  
  | return false  
end  
if ( $\varphi, \alpha$ ) =  $\top$  then  
  | return true  
end  
choose an unassigned  $x \in \mathcal{V}(\varphi)$ ;  
if  $DLL(\varphi, \alpha \cup [\varphi \leftarrow 0])$  then  
  | return true  
else  
  | return  $DLL(\varphi, \alpha \cup [\varphi \leftarrow 1])$   
end
```

**TODO:** How to choose variables? Can we do reductions? Further improvements?

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem

Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# DLL Algorithm

## DLL Algorithm after Davis, Logeman, Loveland

### DLL

**Input:** Formula  $\varphi$  in Propositional Logic, interpretation  $\alpha$

```
DLL( $\varphi, \alpha$ ) if ( $\varphi, \alpha$ ) =  $\perp$  then
  |   return false
end
if ( $\varphi, \alpha$ ) =  $\top$  then
  |   return true
end
choose an unassigned  $x \in \mathcal{V}(\varphi)$ ;
if DLL( $\varphi, \alpha \cup [\varphi \leftarrow 0]$ ) then
  |   return true
else
  |   return DLL( $\varphi, \alpha \cup [\varphi \leftarrow 1]$ )
end
```

**TODO:** How to choose variables? Can we do reductions? Further improvements?

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem

Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# Unit Propagation

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

**Idea:** If in an unsatisfied clause only one literal is still unassigned, the value of this literal can be assigned.

### Example (Unit Propagation)

$$\varphi := (a \vee b \vee c) \wedge (\neg a \vee \neg b)$$

- $\alpha = [a \leftarrow \top]$
  - Because of  $(\neg a \vee \neg b)$ :  $b$  must be assigned to  $\perp$
  - $(\neg a \vee \neg b)$  *got unit*
- 
- Modern solvers spend about 90% of time in Unit Propagation
  - An efficient UP algorithm is the key to a fast SAT solver

# Unit Propagation

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

**Idea:** If in an unsatisfied clause only one literal is still unassigned, the value of this literal can be assigned.

### Example (Unit Propagation)

$$\varphi := (a \vee b \vee c) \wedge (\neg a \vee \neg b)$$

- $\alpha = [a \leftarrow \top]$
- Because of  $(\neg a \vee \neg b)$ :  $b$  must be assigned to  $\perp$
- $(\neg a \vee \neg b)$  *got unit*

- Modern solvers spend about 90% of time in Unit Propagation
- An efficient UP algorithm is the key to a fast SAT solver

# Unit Propagation

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

**Idea:** If in an unsatisfied clause only one literal is still unassigned, the value of this literal can be assigned.

### Example (Unit Propagation)

$$\varphi := (a \vee b \vee c) \wedge (\neg a \vee \neg b)$$

- $\alpha = [a \leftarrow \top]$
  - Because of  $(\neg a \vee \neg b)$ :  $b$  must be assigned to  $\perp$
  - $(\neg a \vee \neg b)$  *got unit*
- 
- Modern solvers spend about 90% of time in Unit Propagation
  - An efficient UP algorithm is the key to a fast SAT solver

# A naive algorithm for UP

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

### A naive algorithm for UP

```
Input: Set of clauses  $C$ , interpretation  $\alpha$   
 $UP(C, \alpha)$  while there are unit clauses in  $C$  do  
    if sole unassigned literal  $a$  is positive then  
        |  $\alpha := \alpha \cup [a \leftarrow \top]$   
    else  
        |  $\alpha := \alpha \cup [a \leftarrow \perp]$   
    end  
end
```

### Problem:

- all clauses must be tested in every step of UP
- ⇒ Exponential blow up

# A naive algorithm for UP

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

### A naive algorithm for UP

```
Input: Set of clauses  $C$ , interpretation  $\alpha$   
 $UP(C, \alpha)$  while there are unit clauses in  $C$  do  
    if sole unassigned literal  $a$  is positive then  
        |  $\alpha := \alpha \cup [a \leftarrow \top]$   
    else  
        |  $\alpha := \alpha \cup [a \leftarrow \perp]$   
    end  
end
```

### Problem:

- all clauses must be tested in every step of UP
- ⇒ Exponential blow up

# Watched Literals

## Generalized SAT Solving

Christoph Zengler

### Introduction

Propositional Logic  
The SAT Problem  
Overview

### Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

### The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

### Parametric Q-SAT

Motivation  
The Algorithm

### Conclusion

### Solution:

- Watch two literals per clause
  - As long as there are two unassigned literals, the clause can not get unit
  - If one of the two watched literals is assigned, search for another
  - If there is no other unassigned literal, the clause is unit with the second watched literal as sole literal
- ⇒ Searching for unit clauses is not longer necessary: After assigning a variable all new unit clauses are calculated

# Watched Literals - Example

## Example (Watched Literals)

$$(a \vee b \vee \neg c \vee d) \wedge (a \vee \neg b \vee \neg c)$$

- watched literals for first clause:  $a, b$
  - watched literals for second clause:  $a, b$
  - $a \leftarrow \perp$
  - new watched literals for first clause:  $b, c$
  - new watched literals for second clause:  $b, c$
  - $c \leftarrow \top$
  - new watched literals for first clause:  $b, d$
  - no further unassigned literal for second clause: got unit with  $b$  sole literal
  - $b \leftarrow \perp$  because of UP
  - no further unassigned literal for first clause: got unit with  $d$  sole literal
  - $d \leftarrow \top$
- ⇒ Satisfying assignment

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# Watched Literals - Example

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

### Example (Watched Literals)

$$(a \vee b \vee \neg c \vee d) \wedge (a \vee \neg b \vee \neg c)$$

- watched literals for first clause:  $a, b$
  - watched literals for second clause:  $a, b$
  - $a \leftarrow \perp$
  - new watched literals for first clause:  $b, c$
  - new watched literals for second clause:  $b, c$
  - $c \leftarrow \top$
  - new watched literals for first clause:  $b, d$
  - no further unassigned literal for second clause: got unit with  $b$  sole literal
  - $b \leftarrow \perp$  because of UP
  - no further unassigned literal for first clause: got unit with  $d$  sole literal
  - $d \leftarrow \top$
- ⇒ Satisfying assignment

# Watched Literals - Example

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

### Example (Watched Literals)

$$(a \vee b \vee \neg c \vee d) \wedge (a \vee \neg b \vee \neg c)$$

- watched literals for first clause:  $a, b$
  - watched literals for second clause:  $a, b$
  - $a \leftarrow \perp$
  - new watched literals for first clause:  $b, c$
  - new watched literals for second clause:  $b, c$
  - $c \leftarrow \top$
  - new watched literals for first clause:  $b, d$
  - no further unassigned literal for second clause: got unit with  $b$  sole literal
  - $b \leftarrow \perp$  because of UP
  - no further unassigned literal for first clause: got unit with  $d$  sole literal
  - $d \leftarrow \top$
- ⇒ Satisfying assignment

# Watched Literals - Example

## Example (Watched Literals)

$$(a \vee b \vee \neg c \vee d) \wedge (a \vee \neg b \vee \neg c)$$

- watched literals for first clause:  $a, b$
  - watched literals for second clause:  $a, b$
  - $a \leftarrow \perp$
  - new watched literals for first clause:  $b, c$
  - new watched literals for second clause:  $b, c$
  - $c \leftarrow \top$
  - new watched literals for first clause:  $b, d$
  - no further unassigned literal for second clause: got unit with  $b$  sole literal
  - $b \leftarrow \perp$  because of UP
  - no further unassigned literal for first clause: got unit with  $d$  sole literal
  - $d \leftarrow \top$
- ⇒ Satisfying assignment

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# Watched Literals - Example

## Example (Watched Literals)

$$(a \vee b \vee \neg c \vee d) \wedge (a \vee \neg b \vee \neg c)$$

- watched literals for first clause:  $a, b$
  - watched literals for second clause:  $a, b$
  - $a \leftarrow \perp$
  - new watched literals for first clause:  $b, c$
  - new watched literals for second clause:  $b, c$
  - $c \leftarrow \top$
  - new watched literals for first clause:  $b, d$
  - no further unassigned literal for second clause: got unit with  $b$  sole literal
  - $b \leftarrow \perp$  because of UP
  - no further unassigned literal for first clause: got unit with  $d$  sole literal
  - $d \leftarrow \top$
- ⇒ Satisfying assignment

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# Watched Literals - Example

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

### Example (Watched Literals)

$$(a \vee b \vee \neg c \vee d) \wedge (a \vee \neg b \vee \neg c)$$

- watched literals for first clause:  $a, b$
  - watched literals for second clause:  $a, b$
  - $a \leftarrow \perp$
  - new watched literals for first clause:  $b, c$
  - new watched literals for second clause:  $b, c$
  - $c \leftarrow \top$
  - new watched literals for first clause:  $b, d$
  - no further unassigned literal for second clause: got unit with  $b$  sole literal
  - $b \leftarrow \perp$  because of UP
  - no further unassigned literal for first clause: got unit with  $d$  sole literal
  - $d \leftarrow \top$
- ⇒ Satisfying assignment

# Watched Literals - Example

## Example (Watched Literals)

$$(a \vee b \vee \neg c \vee d) \wedge (a \vee \neg b \vee \neg c)$$

- watched literals for first clause:  $a, b$
  - watched literals for second clause:  $a, b$
  - $a \leftarrow \perp$
  - new watched literals for first clause:  $b, c$
  - new watched literals for second clause:  $b, c$
  - $c \leftarrow \top$
  - new watched literals for first clause:  $b, d$
  - no further unassigned literal for second clause: got unit with  $b$  sole literal
  - $b \leftarrow \perp$  because of UP
  - no further unassigned literal for first clause: got unit with  $d$  sole literal
  - $d \leftarrow \top$
- ⇒ Satisfying assignment

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# Watched Literals - Example

## Example (Watched Literals)

$$(a \vee b \vee \neg c \vee d) \wedge (a \vee \neg b \vee \neg c)$$

- watched literals for first clause:  $a, b$
  - watched literals for second clause:  $a, b$
  - $a \leftarrow \perp$
  - new watched literals for first clause:  $b, c$
  - new watched literals for second clause:  $b, c$
  - $c \leftarrow \top$
  - new watched literals for first clause:  $b, d$
  - no further unassigned literal for second clause: got unit with  $b$  sole literal
  - $b \leftarrow \perp$  because of UP
  - no further unassigned literal for first clause: got unit with  $d$  sole literal
  - $d \leftarrow \top$
- ⇒ Satisfying assignment

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# Counting Heuristics

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics

Conflict Driven  
Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

**Question:** How to choose the next branch (variable)

**Answer:** Heuristics

### Dynamic Largest Combined Sum (DLCS)

- For each variable: Count the sum of positive and negative occurrences in unsatisfied clauses
- Take the variable with the largest sum
- assign it to  $\top$  if it has more positive occurrences than negatives,  $\perp$  else

### Dynamic Largest Individual Sum (DLIS)

- For each variable: Count the positive and negative occurrences in unsatisfied clauses separately
  - Take the variable with the largest individual sum
  - assign it to  $\top$  if it's from a positive sum,  $\perp$  else
- 
- Both heuristics can also be randomized

# Counting Heuristics

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics

Conflict Driven  
Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

**Question:** How to choose the next branch (variable)

**Answer:** Heuristics

### Dynamic Largest Combined Sum (DLCS)

- For each variable: Count the sum of positive and negative occurrences in unsatisfied clauses
- Take the variable with the largest sum
- assign it to  $\top$  if it has more positive occurrences than negatives,  $\perp$  else

### Dynamic Largest Individual Sum (DLIS)

- For each variable: Count the positive and negative occurrences in unsatisfied clauses separately
  - Take the variable with the largest individual sum
  - assign it to  $\top$  if it's from a positive sum,  $\perp$  else
- 
- Both heuristics can also be randomized

# Counting Heuristics

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics

Conflict Driven  
Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

**Question:** How to choose the next branch (variable)

**Answer:** Heuristics

### Dynamic Largest Combined Sum (DLCS)

- For each variable: Count the sum of positive and negative occurrences in unsatisfied clauses
- Take the variable with the largest sum
- assign it to  $\top$  if it has more positive occurrences than negatives,  $\perp$  else

### Dynamic Largest Individual Sum (DLIS)

- For each variable: Count the positive and negative occurrences in unsatisfied clauses separately
  - Take the variable with the largest individual sum
  - assign it to  $\top$  if it's from a positive sum,  $\perp$  else
- 
- Both heuristics can also be randomized

# Counting Heuristics

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics

Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

**Question:** How to choose the next branch (variable)

**Answer:** Heuristics

## Dynamic Largest Combined Sum (DLCS)

- For each variable: Count the sum of positive and negative occurrences in unsatisfied clauses
- Take the variable with the largest sum
- assign it to  $\top$  if it has more positive occurrences than negatives,  $\perp$  else

## Dynamic Largest Individual Sum (DLIS)

- For each variable: Count the positive and negative occurrences in unsatisfied clauses separately
  - Take the variable with the largest individual sum
  - assign it to  $\top$  if it's from a positive sum,  $\perp$  else
- Both heuristics can also be randomized

# Counting Heuristics

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics

Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

**Question:** How to choose the next branch (variable)

**Answer:** Heuristics

## Dynamic Largest Combined Sum (DLCS)

- For each variable: Count the sum of positive and negative occurrences in unsatisfied clauses
- Take the variable with the largest sum
- assign it to  $\top$  if it has more positive occurrences than negatives,  $\perp$  else

## Dynamic Largest Individual Sum (DLIS)

- For each variable: Count the positive and negative occurrences in unsatisfied clauses separately
  - Take the variable with the largest individual sum
  - assign it to  $\top$  if it's from a positive sum,  $\perp$  else
- 
- Both heuristics can also be randomized

# MOM Heuristics

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics

Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

## MOM Heuristic

**M**aximal **O**ccurrence in clauses of **M**inimal size

- maximum number of Unit Propagations in the next step
- $numpos_l$ ,  $numneg_l$  number of pos. and neg. occurrences in clauses of minimal size  $l$
- calculation:

$$(numpos_l + numneg_l) \cdot 2^k + (numpos_l \cdot numneg_l)$$

with  $k$  a number to simulate lexicographic order

## Improvement in REDLOG: ZMOM

- Calculate MOM value from all clauses, not just clauses of minimal size
- choose variable with maximum MOM value and occurrence in at least one clause of minimal size
- Advantage: calculation can be implemented more efficiently
- Showed best performance

# MOM Heuristics

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics

Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

## MOM Heuristic

### Maximal Occurrence in clauses of Minimal size

- maximum number of Unit Propagations in the next step
- $numpos_l$ ,  $numneg_l$  number of pos. and neg. occurrences in clauses of minimal size  $l$
- calculation:

$$(numpos_l + numneg_l) \cdot 2^k + (numpos_l \cdot numneg_l)$$

with  $k$  a number to simulate lexicographic order

### Improvement in REDLOG: ZMOM

- Calculate MOM value from all clauses, not just clauses of minimal size
- choose variable with maximum MOM value and occurrence in at least one clause of minimal size
- Advantage: calculation can be implemented more efficiently
- Showed best performance

# Activity Heuristic / Benchmarks

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic

The SAT Problem

Overview

Efficient SAT  
Solving

Unit Propagation

Branching Heuristics

Conflict Driven

Clause Learning

Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem

An Algorithm for

Q-SAT

Benchmarks

Parametric  
Q-SAT

Motivation

The Algorithm

Conclusion

## Activity Heuristic

Store an activity value for each variable:

- increase the value if the variable occurs in a conflict
- decrease the value from time to time

⇒ the chosen variable currently caused many conflicts

Benchmarks:

Class	# vars	# cl	ZMOM	Act	DLCS
aim-1_6-no	200	320	<b>0.11</b>	0.27	<b>0.11</b>
aim-2_0-no	200	400	<b>0.11</b>	0.5	0.14
aim-1_6-yes1	200	320	16.26	<b>0.83</b>	14.78
aim-2_0-yes1	200	400	<b>1.49</b>	6.6	2.04
aim-3_4-yes1	200	680	19.08	37.46	<b>14.03</b>
aim-6_0-yes1	200	1200	6.81	<b>5.89</b>	18.73
ii8	-1068	-8214	<b>8.54</b>	20.25	12.16
ii16	-1728	-24792	<b>187.65</b>	657.56	264.54

# Activity Heuristic / Benchmarks

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics

Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

## Activity Heuristic

Store an activity value for each variable:

- increase the value if the variable occurs in a conflict
- decrease the value from time to time

⇒ the chosen variable currently caused many conflicts

Benchmarks:

Class	# vars	# cl	ZMOM	Act	DLCS
aim-1_6-no	200	320	<b>0.11</b>	0.27	<b>0.11</b>
aim-2_0-no	200	400	<b>0.11</b>	0.5	0.14
aim-1_6-yes1	200	320	16.26	<b>0.83</b>	14.78
aim-2_0-yes1	200	400	<b>1.49</b>	6.6	2.04
aim-3_4-yes1	200	680	19.08	37.46	<b>14.03</b>
aim-6_0-yes1	200	1200	6.81	<b>5.89</b>	18.73
ii8	-1068	-8214	<b>8.54</b>	20.25	12.16
ii16	-1728	-24792	<b>187.65</b>	657.56	264.54

# Conflict Driven Clause Learning (CDCL) - 1

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

Problem with DLL: we forget information we already computed

### Example (Situation with DLL)

$$\varphi := \bigwedge_{i=0}^n \psi_i \wedge (x \vee y \vee z) \wedge (\neg x \vee \neg y) \wedge (\neg x \vee z) \wedge (y \vee \neg z)$$

- $\psi_i$  further clauses
  - first  $m$  branching variables:  $a_1, \dots, a_m$
  - $m + 1$ -th branching variable:  $x$
  - if  $x \leftarrow \top$  then UP:  $y \leftarrow \perp, z \leftarrow \top \implies$  Empty clause:  $(y \vee \neg z)$
- $\Rightarrow$   $x$  must be assigned to  $\perp$
- But after a backtrack to variables  $a_1, \dots, a_m$  we “forget” this information

# CDCL - 2

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

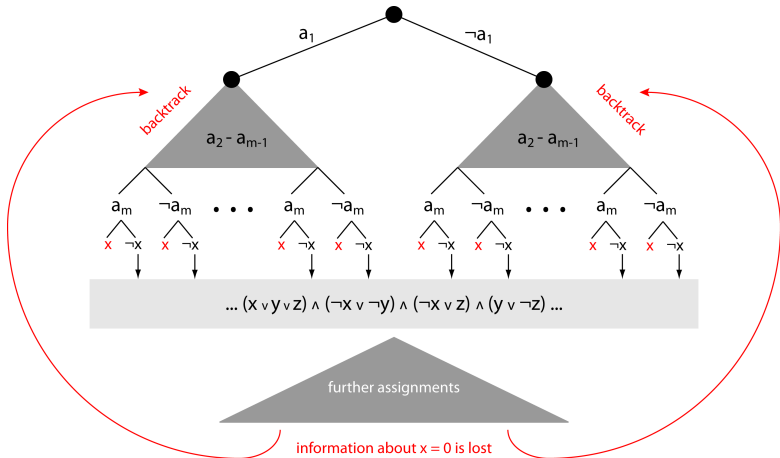
## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion



**Idea:** Learn new clauses during backtracking procedure

## Resolution

Resolution of two clauses in a CNF

- $\varphi := A \vee p$
- $\psi := B \vee \neg p$
- $A, B$  disjunctions of other literals  $x_i$

is defined as:

- $\varphi \oplus \psi = A \vee B$

This does not change the satisfiability of the formula  
 $\implies$  redundant in respect to the original formula

After a conflict: Use of resolution to learn a new clause

**Idea:** Learn new clauses during backtracking procedure

## Resolution

Resolution of two clauses in a CNF

- $\varphi := A \vee p$
- $\psi := B \vee \neg p$
- $A, B$  disjunctions of other literals  $x_i$

is defined as:

- $\varphi \oplus \psi = A \vee B$

This does not change the satisfiability of the formula  
 $\implies$  redundant in respect to the original formula

After a conflict: Use of resolution to learn a new clause

**Idea:** Learn new clauses during backtracking procedure

## Resolution

Resolution of two clauses in a CNF

- $\varphi := A \vee p$
- $\psi := B \vee \neg p$
- $A, B$  disjunctions of other literals  $x_i$

is defined as:

- $\varphi \oplus \psi = A \vee B$

This does not change the satisfiability of the formula  
 $\implies$  redundant in respect to the original formula

After a conflict: Use of resolution to learn a new clause

# CDCL - The Algorithm

**Input:** Formula  $f$  in Propositional Logic

level := 0;

$\alpha := \emptyset$ ;

**while** true **do**

UnitPropagation( $f, \alpha$ );

**if** a conflict is reached **then**

ec := empty clause;

level := analyseConflict(ec, C (the set of clauses));

**if** level = 0 **then**

| **return** false

**end**

backtrack(level);

**else**

**if** formula is satisfied **then**

| **return** true

**end**

level := level + 1;

choose an unassigned  $x \in \mathcal{V}(F)$ ;

$\alpha := \alpha \cup [x \leftarrow 0]$ ;

**end**

**end**

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# CDCL - The Conflict Analysis

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

- we store a *reason* for each assigned variable
- Why it was assigned?
  - decision variable (because of a branch): *nil*
  - implied variable (because of UP): the corresponding unit clause

### The Analysis of a Conflict

```
Input: an empty clause ec, the set of clauses C  
lv := the variable assigned last;  
reas := the reason of lv;  
newclause := resolve(ec,reas);  
while the stop criterion for newclause is not met do  
    | cv := chooseLiteral(newclause);  
    | reas := the reason of cv;  
    | newclause := resolve(newclause,reas);  
end  
C := C ∪ {newclause};  
level := calculateBTLevel(newclause);  
return level;
```

# CDCL - The Conflict Analysis

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

- we store a *reason* for each assigned variable
- Why it was assigned?
  - decision variable (because of a branch): *nil*
  - implied variable (because of UP): the corresponding unit clause

### The Analysis of a Conflict

```
Input: an empty clause ec, the set of clauses C  
lv := the variable assigned last;  
reas := the reason of lv;  
newclause := resolve(ec,reas);  
while the stop criterion for newclause is not met do  
    | cv := chooseLiteral(newclause);  
    | reas := the reason of cv;  
    | newclause := resolve(newclause,reas);  
end  
C := C ∪ {newclause};  
level := calculateBTLevel(newclause);  
return level;
```

# CDCL - What is the stop criterion

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

When do we stop with resolution and learn a new clause?

- If only one literal in the new clause is at the current decision level

What is the new backtrack level?

- It is the greatest level of a variable in the new clause really smaller than the current decision level

### Example

New learnt clause if the new clause is

$$(a_{(2)} \vee \neg b_{(1)} \vee h_{(3)} \vee k_{(3)} \vee \neg l_{(5)})$$

we stop.

- Only one literal at highest level ( $l$  at level 5)
- Backtrack to level 3

Why? After backtracking the new learnt clause is a unit clause and so we get a new UP.

# CDCL - What is the stop criterion

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

When do we stop with resolution and learn a new clause?

- If only one literal in the new clause is at the current decision level

What is the new backtrack level?

- It is the greatest level of a variable in the new clause really smaller than the current decision level

### Example

New learnt clause if the new clause is

$$(a_{(2)} \vee \neg b_{(1)} \vee h_{(3)} \vee k_{(3)} \vee \neg l_{(5)})$$

we stop.

- Only one literal at highest level ( $l$  at level 5)
- Backtrack to level 3

**Why?** After backtracking the new learnt clause is a unit clause and so we get a new UP.

# CDCL - What is the stop criterion

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

When do we stop with resolution and learn a new clause?

- If only one literal in the new clause is at the current decision level

What is the new backtrack level?

- It is the greatest level of a variable in the new clause really smaller than the current decision level

## Example

New learnt clause If the new clause is

$$(a_{(2)} \vee \neg b_{(1)} \vee h_{(3)} \vee k_{(3)} \vee \neg l_{(5)})$$

we stop.

- Only one literal at highest level ( $l$  at level 5)
- Backtrack to level 3

Why? After backtracking the new learnt clause is a unit clause and so we get a new UP.

# CDCL - What is the stop criterion

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

When do we stop with resolution and learn a new clause?

- If only one literal in the new clause is at the current decision level

What is the new backtrack level?

- It is the greatest level of a variable in the new clause really smaller than the current decision level

## Example

New learnt clause If the new clause is

$$(a_{(2)} \vee \neg b_{(1)} \vee h_{(3)} \vee k_{(3)} \vee \neg l_{(5)})$$

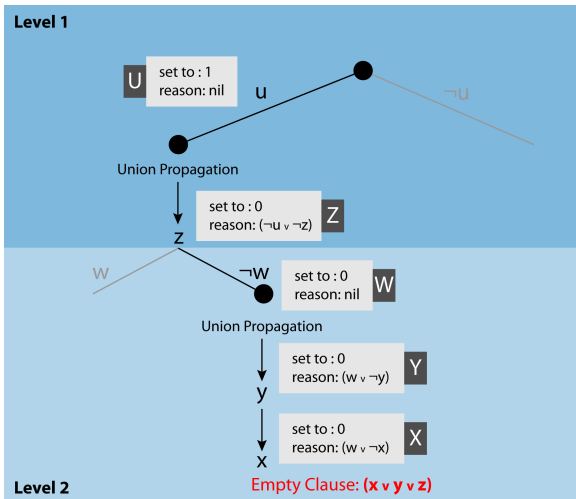
we stop.

- Only one literal at highest level ( $l$  at level 5)
- Backtrack to level 3

**Why?** After backtracking the new learnt clause is a unit clause and so we get a new UP.

# CDCL - Example - 1

$$\varphi := (u \vee \neg w) \wedge (\neg u \vee \neg z) \wedge (w \vee \neg y) \wedge (w \vee \neg x) \wedge (x \vee y \vee z)$$



Generalized SAT Solving

Christoph Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

Parametric Q-SAT

Motivation  
The Algorithm

Conclusion

# CDCL - Example - 2

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

$$\varphi := (u \vee \neg w) \wedge (\neg u \vee \neg z) \wedge (w \vee \neg y) \wedge (w \vee \neg x) \wedge (x \vee y \vee z)$$

- $u$  assigned to  $\top$  at level 1, reason *nil*
- $z$  assigned to  $\perp$  at level 1, reason  $\neg u \vee \neg z$
- $w$  assigned to  $\perp$  at level 2, reason *nil*
- $y$  assigned to  $\perp$  at level 2, reason  $w \vee \neg y$
- $x$  assigned to  $\perp$  at level 2, reason  $w \vee \neg x$
- **empty clause:  $(x \vee y \vee z)$**

### Example (Learning progress)

- Start with empty clause and the reason of the last assigned variable:  
 $(x_{(2)} \vee y_{(2)} \vee z_{(1)}) \oplus (w_{(2)} \vee \neg x_{(2)}) = (y_{(2)} \vee z_{(1)} \vee w_{(2)})$
- $(y_{(2)} \vee z_{(1)} \vee w_{(2)}) \oplus (w_{(2)} \vee \neg y_{(2)}) = (w_{(2)} \vee z_{(1)})$
- new learnt clause:  $(w \vee z)$
- backtrack to level 1

# CDCL - Example - 2

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

$$\varphi := (u \vee \neg w) \wedge (\neg u \vee \neg z) \wedge (w \vee \neg y) \wedge (w \vee \neg x) \wedge (x \vee y \vee z)$$

- $u$  assigned to  $\top$  at level 1, reason *nil*
- $z$  assigned to  $\perp$  at level 1, reason  $\neg u \vee \neg z$
- $w$  assigned to  $\perp$  at level 2, reason *nil*
- $y$  assigned to  $\perp$  at level 2, reason  $w \vee \neg y$
- $x$  assigned to  $\perp$  at level 2, reason  $w \vee \neg x$
- **empty clause:  $(x \vee y \vee z)$**

### Example (Learning progress)

- Start with empty clause and the reason of the last assigned variable:  
 $(x_{(2)} \vee y_{(2)} \vee z_{(1)}) \oplus (w_{(2)} \vee \neg x_{(2)}) = (y_{(2)} \vee z_{(1)} \vee w_{(2)})$
- $(y_{(2)} \vee z_{(1)} \vee w_{(2)}) \oplus (w_{(2)} \vee \neg y_{(2)}) = (w_{(2)} \vee z_{(1)})$
- new learnt clause:  $(w \vee z)$
- backtrack to level 1

# CDCL - Example - 2

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

$$\varphi := (u \vee \neg w) \wedge (\neg u \vee \neg z) \wedge (w \vee \neg y) \wedge (w \vee \neg x) \wedge (x \vee y \vee z)$$

- $u$  assigned to  $\top$  at level 1, reason *nil*
- $z$  assigned to  $\perp$  at level 1, reason  $\neg u \vee \neg z$
- $w$  assigned to  $\perp$  at level 2, reason *nil*
- $y$  assigned to  $\perp$  at level 2, reason  $w \vee \neg y$
- $x$  assigned to  $\perp$  at level 2, reason  $w \vee \neg x$
- **empty clause:  $(x \vee y \vee z)$**

### Example (Learning progress)

- Start with empty clause and the reason of the last assigned variable:  
 $(x_{(2)} \vee y_{(2)} \vee z_{(1)}) \oplus (w_{(2)} \vee \neg x_{(2)}) = (y_{(2)} \vee z_{(1)} \vee w_{(2)})$
- $(y_{(2)} \vee z_{(1)} \vee w_{(2)}) \oplus (w_{(2)} \vee \neg y_{(2)}) = (w_{(2)} \vee z_{(1)})$
- new learnt clause:  $(w \vee z)$
- backtrack to level 1

# CDCL - Example - 2

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

$$\varphi := (u \vee \neg w) \wedge (\neg u \vee \neg z) \wedge (w \vee \neg y) \wedge (w \vee \neg x) \wedge (x \vee y \vee z)$$

- $u$  assigned to  $\top$  at level 1, reason *nil*
- $z$  assigned to  $\perp$  at level 1, reason  $\neg u \vee \neg z$
- $w$  assigned to  $\perp$  at level 2, reason *nil*
- $y$  assigned to  $\perp$  at level 2, reason  $w \vee \neg y$
- $x$  assigned to  $\perp$  at level 2, reason  $w \vee \neg x$
- **empty clause:  $(x \vee y \vee z)$**

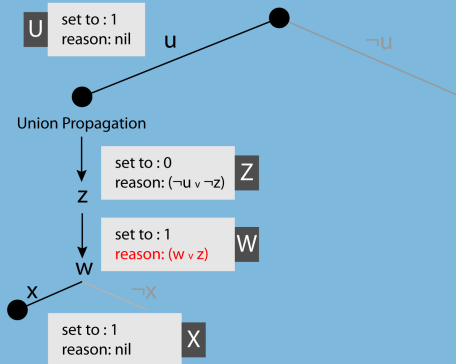
### Example (Learning progress)

- Start with empty clause and the reason of the last assigned variable:  
 $(x_{(2)} \vee y_{(2)} \vee z_{(1)}) \oplus (w_{(2)} \vee \neg x_{(2)}) = (y_{(2)} \vee z_{(1)} \vee w_{(2)})$
- $(y_{(2)} \vee z_{(1)} \vee w_{(2)}) \oplus (w_{(2)} \vee \neg y_{(2)}) = (w_{(2)} \vee z_{(1)})$
- new learnt clause:  $(w \vee z)$
- backtrack to level 1

# CDCL - Example - 3

$$\varphi := (u \vee \neg w) \wedge (\neg u \vee \neg z) \wedge (w \vee \neg y) \wedge (w \vee \neg x) \wedge (x \vee y \vee z) \wedge (w \vee z)$$

## Level 1



Generalized SAT Solving

Christoph Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

Parametric Q-SAT

Motivation  
The Algorithm

Conclusion

# Further Improvements

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

### Restarts

- Restart after a certain number of learnt clauses
- Increase this number with time

Avoid that early wrong branches have too much impact

### Clause deletion

- like activity heuristics: store activity of clauses
- delete inactive clauses from time to time

Avoid blow up of number of clauses

### Simplifications

- Delete clauses with only one literal
- Set this literal and delete all satisfied clauses

# Further Improvements

## Generalized SAT Solving

Christoph Zengler

### Introduction

Propositional Logic  
The SAT Problem  
Overview

### Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

### The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

### Parametric Q-SAT

Motivation  
The Algorithm

### Conclusion

#### Restarts

- Restart after a certain number of learnt clauses
- Increase this number with time

Avoid that early wrong branches have too much impact

#### Clause deletion

- like activity heuristics: store activity of clauses
- delete inactive clauses from time to time

Avoid blow up of number of clauses

#### Simplifications

- Delete clauses with only one literal
- Set this literal and delete all satisfied clauses

# Further Improvements

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

### Restarts

- Restart after a certain number of learnt clauses
- Increase this number with time

Avoid that early wrong branches have too much impact

### Clause deletion

- like activity heuristics: store activity of clauses
- delete inactive clauses from time to time

Avoid blow up of number of clauses

### Simplifications

- Delete clauses with only one literal
- Set this literal and delete all satisfied clauses

# Comparison to QE

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

Benchmark	QE	QSAT
aim-50-1_6-no	13.17	0.01
aim-50-1_6-yes1	3.14	0.02
i18	3230.04 (53 min)	8.53

- QE is basically DLL with some simplifications

## Example

### Processor verification

- Correct verification of a 1-issue, 5-stage DLX processor, 6 instruction types: register-register, register-immediate, load, store, branch, jump in **7 seconds** (295 variables, 1592 clauses)
- Finding of a failure in a DLX processor with 2 pipelines in **5 seconds** (1506 variables, 12763 clauses)

# Comparison to QE

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

Benchmark	QE	QSAT
aim-50-1_6-no	13.17	0.01
aim-50-1_6-yes1	3.14	0.02
i18	3230.04 (53 min)	8.53

- QE is basically DLL with some simplifications

### Example

#### Processor verification

- Correct verification of a 1-issue, 5-stage DLX processor, 6 instruction types: register-register, register-immediate, load, store, branch, jump in **7 seconds** (295 variables, 1592 clauses)
- Finding of a failure in a DLX processor with 2 pipelines in **5 seconds** (1506 variables, 12763 clauses)

# Comparison to QE

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

Benchmark	QE	QSAT
aim-50-1_6-no	13.17	0.01
aim-50-1_6-yes1	3.14	0.02
i18	3230.04 (53 min)	8.53

- QE is basically DLL with some simplifications

## Example

### Processor verification

- Correct verification of a 1-issue, 5-stage DLX processor, 6 instruction types: register-register, register-immediate, load, store, branch, jump in **7 seconds** (295 variables, 1592 clauses)
- Finding of a failure in a DLX processor with 2 pipelines in **5 seconds** (1506 variables, 12763 clauses)

# Introduction

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

Till now we can solve formulas over Propositional Logic  
This is equivalent to solve formulas in First-Order Logic with only existential variables.

### Example

The SAT Problem

$$(a \vee b \vee c) \wedge (a \vee \neg b) \wedge (\neg b \vee \neg c)$$

can also be written as:

$$\exists a \exists b \exists c ((a = T \vee b = T \vee c = T) \wedge (a = T \vee \neg b = T) \wedge (\neg b = T \vee \neg c = T))$$

**What we want now:** Expand our methods for formulas with existential and universal variables

# Introduction

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

Till now we can solve formulas over Propositional Logic  
This is equivalent to solve formulas in First-Order Logic with only existential variables.

### Example

The SAT Problem

$$(a \vee b \vee c) \wedge (a \vee \neg b) \wedge (\neg b \vee \neg c)$$

can also be written as:

$$\exists a \exists b \exists c ((a = \top \vee b = \top \vee c = \top) \wedge (a = \top \vee \neg b = \top) \wedge (\neg b = \top \vee \neg c = \top))$$

**What we want now:** Expand our methods for formulas with existential and universal variables

# Introduction

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

Till now we can solve formulas over Propositional Logic  
This is equivalent to solve formulas in First-Order Logic with only existential variables.

### Example

The SAT Problem

$$(a \vee b \vee c) \wedge (a \vee \neg b) \wedge (\neg b \vee \neg c)$$

can also be written as:

$$\exists a \exists b \exists c ((a = \top \vee b = \top \vee c = \top) \wedge (a = \top \vee \neg b = \top) \wedge (\neg b = \top \vee \neg c = \top))$$

**What we want now:** Expand our methods for formulas with existential and universal variables

# The Q-SAT Problem

## Q-SAT

The Quantified Boolean Satisfiability Problem is the question, whether a given quantified input formula  $\varphi$  is *true* or *false*.

## Example (Q-SAT problems)

$$\forall x \exists y ((x \vee y) \wedge (\neg x \vee \neg y)) = \top$$

$$\forall x \forall y ((x \vee y) \wedge (\neg x \vee \neg y)) = \perp$$

- Formula  $\varphi$  must be in prenex normal form
- Group quantifiers of same type to one quantification level
- Heuristics must obey quantification level

## Example

$$\underbrace{\exists x \exists y}_{\text{level 1}} \underbrace{\forall z \forall w}_{\text{level 2}} \underbrace{\exists u}_{\text{level 3}} (x \vee y \vee z \vee w \vee u)$$

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# The Q-SAT Problem

## Q-SAT

The Quantified Boolean Satisfiability Problem is the question, whether a given quantified input formula  $\varphi$  is *true* or *false*.

## Example (Q-SAT problems)

$$\forall x \exists y ((x \vee y) \wedge (\neg x \vee \neg y)) = \top$$

$$\forall x \forall y ((x \vee y) \wedge (\neg x \vee \neg y)) = \perp$$

- Formula  $\varphi$  must be in prenex normal form
- Group quantifiers of same type to one quantification level
- Heuristics must obey quantification level

## Example

$$\underbrace{\exists x \exists y}_{\text{level 1}} \underbrace{\forall z \forall w}_{\text{level 2}} \underbrace{\exists u}_{\text{level 3}} (x \vee y \vee z \vee w \vee u)$$

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# The Q-SAT Problem

## Q-SAT

The Quantified Boolean Satisfiability Problem is the question, whether a given quantified input formula  $\varphi$  is *true* or *false*.

## Example (Q-SAT problems)

$$\forall x \exists y ((x \vee y) \wedge (\neg x \vee \neg y)) = \top$$

$$\forall x \forall y ((x \vee y) \wedge (\neg x \vee \neg y)) = \perp$$

- Formula  $\varphi$  must be in prenex normal form
- Group quantifiers of same type to one quantification level
- Heuristics must obey quantification level

## Example

$$\underbrace{\exists x \exists y}_{\text{level 1}} \underbrace{\forall z \forall w}_{\text{level 2}} \underbrace{\exists u}_{\text{level 3}} (x \vee y \vee z \vee w \vee u)$$

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# The Q-SAT Problem

## Q-SAT

The Quantified Boolean Satisfiability Problem is the question, whether a given quantified input formula  $\varphi$  is *true* or *false*.

## Example (Q-SAT problems)

$$\forall x \exists y ((x \vee y) \wedge (\neg x \vee \neg y)) = \top$$

$$\forall x \forall y ((x \vee y) \wedge (\neg x \vee \neg y)) = \perp$$

- Formula  $\varphi$  must be in prenex normal form
- Group quantifiers of same type to one quantification level
- Heuristics must obey quantification level

## Example

$$\underbrace{\exists x \exists y}_{\text{level 1}} \underbrace{\forall z \forall w}_{\text{level 2}} \underbrace{\exists u}_{\text{level 3}} (x \vee y \vee z \vee w \vee u)$$

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# The Q-SAT Problem

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

## Q-SAT

The Quantified Boolean Satisfiability Problem is the question, whether a given quantified input formula  $\varphi$  is *true* or *false*.

## Example (Q-SAT problems)

$$\forall x \exists y ((x \vee y) \wedge (\neg x \vee \neg y)) = \top$$

$$\forall x \forall y ((x \vee y) \wedge (\neg x \vee \neg y)) = \perp$$

- Formula  $\varphi$  must be in prenex normal form
- Group quantifiers of same type to one quantification level
- Heuristics must obey quantification level

## Example

$$\underbrace{\exists x \exists y}_{\text{level1}} \underbrace{\forall z \forall w}_{\text{level2}} \underbrace{\exists u}_{\text{level3}} (x \vee y \vee z \vee w \vee u)$$

# Complexity of Q-SAT

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

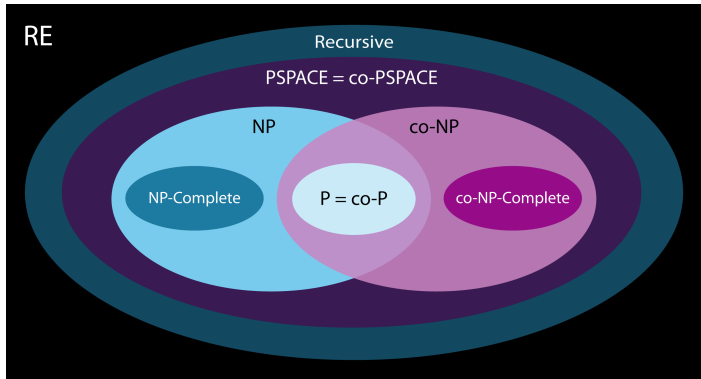
## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion



# Complexity of Q-SAT

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

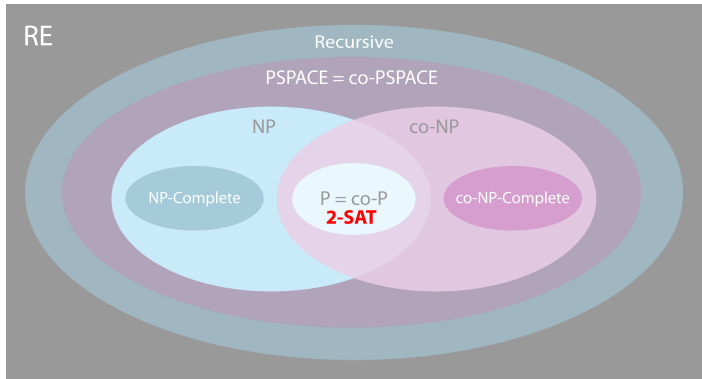
## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion



# Complexity of Q-SAT

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

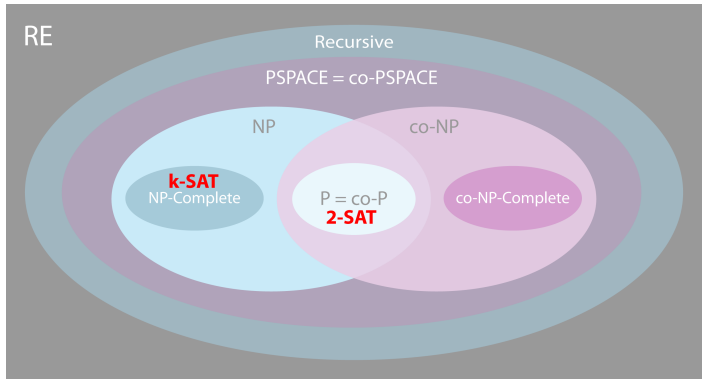
## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion



# Complexity of Q-SAT

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

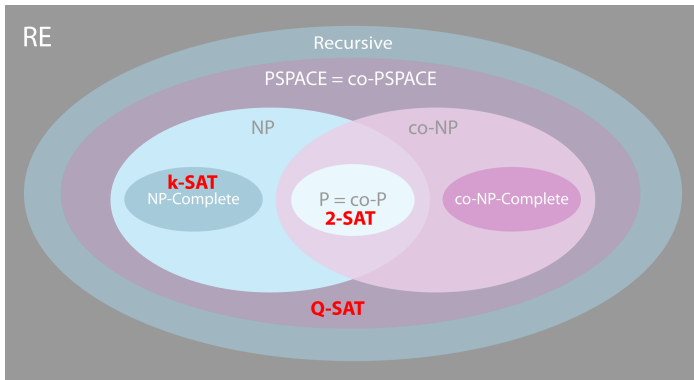
## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion



# An extended search-tree

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

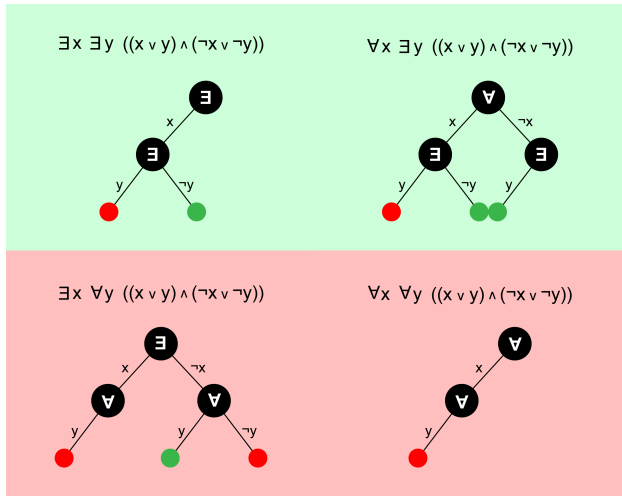


Figure: Search tree for a Q-SAT problem

# An extended algorithm for Q-SAT

- Backtracking not only in conflict cases but also for universal nodes

## Q-SAT Algorithm

```
level := 1;
while true do
  choose an unassigned  $x \in \mathcal{V}(F)$  (wrt. the q-level);
   $\alpha := \alpha \cup [x \leftarrow 0]$ ;
  while true do
    UnitPropagation( $f, \alpha$ );
    if a conflict is reached then
      level := analyseConflict();
      if level = 0 then
        return false
      end
      backtrack(level);
    else
      if formula is satisfied then
        level := analyseSAT();
        if level = 0 then
          return true
        end
        backtrack(level)
      else
        level := level + 1;
      end
    end
  end
end
```

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# New Rules

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

- $E(C)$  the existential literals of a clause  $C$
- $U(C)$  the universal literals of a clause  $C$
- $ql(l)$  the quantification level of a literal  $l$

### Unit clauses

A clause  $C$  is unit if

- 1 it exists  $e \in E(C)$ ,  $e = nil$ .
- 2 For any  $e' \in E(C)$ ,  $e' \neq e$ :  $e' = \perp$
- 3 for all  $u \in U(C)$  with  $u \neq \top$ :  $u = nil \Rightarrow ql(u) > ql(e)$

### Example (Unit Clause)

$a, b, c$  are existential,  $x, y$  are universal variables

[ $a \leftarrow \perp, c \leftarrow \top, x \leftarrow \perp$ ] we are currently at level 5

- $(a_{(2)} \vee b_{(5)} \vee \neg c_{(3)} \vee x_{(1)} \vee y_{(6)})$  is unit
- $(a_{(2)} \vee b_{(5)} \vee \neg c_{(3)} \vee x_{(4)} \vee y_{(1)})$  is not unit

# New Rules

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

- $E(C)$  the existential literals of a clause  $C$
- $U(C)$  the universal literals of a clause  $C$
- $ql(l)$  the quantification level of a literal  $l$

### Unit clauses

A clause  $C$  is unit if

- 1 it exists  $e \in E(C)$ ,  $e = nil$ .
- 2 For any  $e' \in E(C)$ ,  $e' \neq e$ :  $e' = \perp$
- 3 for all  $u \in U(C)$  with  $u \neq \top$ :  $u = nil \Rightarrow ql(u) > ql(e)$

### Example (Unit Clause)

$a, b, c$  are existential,  $x, y$  are universal variables

$[a \leftarrow \perp, c \leftarrow \top, x \leftarrow \perp]$  we are currently at level 5

- $(a_{(2)} \vee b_{(5)} \vee \neg c_{(3)} \vee x_{(1)} \vee y_{(6)})$  is unit
- $(a_{(2)} \vee b_{(5)} \vee \neg c_{(3)} \vee x_{(4)} \vee y_{(1)})$  is not unit

# New Rules

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

- $E(C)$  the existential literals of a clause  $C$
- $U(C)$  the universal literals of a clause  $C$
- $ql(l)$  the quantification level of a literal  $l$

### Unit clauses

A clause  $C$  is unit if

- 1 it exists  $e \in E(C)$ ,  $e = nil$ .
- 2 For any  $e' \in E(C)$ ,  $e' \neq e$ :  $e' = \perp$
- 3 for all  $u \in U(C)$  with  $u \neq \top$ :  $u = nil \Rightarrow ql(u) > ql(e)$

### Example (Unit Clause)

$a, b, c$  are existential,  $x, y$  are universal variables

[ $a \leftarrow \perp, c \leftarrow \top, x \leftarrow \perp$ ] we are currently at level 5

- $(a_{(2)} \vee b_{(5)} \vee \neg c_{(3)} \vee x_{(1)} \vee y_{(6)})$  is unit
- $(a_{(2)} \vee b_{(5)} \vee \neg c_{(3)} \vee x_{(4)} \vee y_{(1)})$  is **not unit**

# New Rules

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

- $E(C)$  the existential literals of a clause  $C$
- $U(C)$  the universal literals of a clause  $C$
- $ql(l)$  the quantification level of a literal  $l$

## Unit clauses

A clause  $C$  is unit if

- 1 it exists  $e \in E(C)$ ,  $e = nil$ .
- 2 For any  $e' \in E(C)$ ,  $e' \neq e$ :  $e' = \perp$
- 3 for all  $u \in U(C)$  with  $u \neq \top$ :  $u = nil \Rightarrow ql(u) > ql(e)$

## Example (Unit Clause)

$a, b, c$  are existential,  $x, y$  are universal variables

[ $a \leftarrow \perp, c \leftarrow \top, x \leftarrow \perp$ ] we are currently at level 5

- $(a_{(2)} \vee b_{(5)} \vee \neg c_{(3)} \vee x_{(1)} \vee y_{(6)})$  is unit
- $(a_{(2)} \vee b_{(5)} \vee \neg c_{(3)} \vee x_{(4)} \vee y_{(1)})$  is **not unit**

# New Rules

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

- $E(C)$  the existential literals of a clause  $C$
- $U(C)$  the universal literals of a clause  $C$
- $ql(l)$  the quantification level of a literal  $l$

### Unit clauses

A clause  $C$  is unit if

- 1 it exists  $e \in E(C)$ ,  $e = nil$ .
- 2 For any  $e' \in E(C)$ ,  $e' \neq e$ :  $e' = \perp$
- 3 for all  $u \in U(C)$  with  $u \neq \top$ :  $u = nil \Rightarrow ql(u) > ql(e)$

### Example (Unit Clause)

$a, b, c$  are existential,  $x, y$  are universal variables

$[a \leftarrow \perp, c \leftarrow \top, x \leftarrow \perp]$  we are currently at level 5

- $(a_{(2)} \vee b_{(5)} \vee \neg c_{(3)} \vee x_{(1)} \vee y_{(6)})$  is unit
- $(a_{(2)} \vee b_{(5)} \vee \neg c_{(3)} \vee x_{(4)} \vee y_{(1)})$  is **not unit**

# Empty clauses

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

When is a clause an empty clause?

### Empty clause

A clause  $C$  is an empty clause if

- 1 for all  $e \in E(C) : e = \perp$
- 2 for all  $u \in U(C) : u \neq \top$

### Example (Empty clause)

$a, b, c$  are existential,  $x, y$  are universal variables

[ $a \leftarrow \perp, b \leftarrow \perp, c \leftarrow \top, x \leftarrow \perp, y \leftarrow \perp$ ] we are currently at level 5

- $(a_{(2)} \vee b_{(5)} \vee \neg c_{(3)} \vee x_{(1)} \vee y_{(6)})$  is an empty clause
- $(a_{(2)} \vee b_{(5)} \vee c_{(3)} \vee x_{(4)} \vee y_{(1)})$  is not an empty clause

# Empty clauses

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

When is a clause an empty clause?

### Empty clause

A clause  $C$  is an empty clause if

- 1 for all  $e \in E(C) : e = \perp$
- 2 for all  $u \in U(C) : u \neq \top$

### Example (Empty clause)

$a, b, c$  are existential,  $x, y$  are universal variables

$[a \leftarrow \perp, b \leftarrow \perp, c \leftarrow \top, x \leftarrow \perp, ]$  we are currently at level 5

- $(a_{(2)} \vee b_{(5)} \vee \neg c_{(3)} \vee x_{(1)} \vee y_{(6)})$  is an empty clause
- $(a_{(2)} \vee b_{(5)} \vee c_{(3)} \vee x_{(4)} \vee y_{(1)})$  is **not an empty clause**

# Empty clauses

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

When is a clause an empty clause?

### Empty clause

A clause  $C$  is an empty clause if

- 1 for all  $e \in E(C) : e = \perp$
- 2 for all  $u \in U(C) : u \neq \top$

### Example (Empty clause)

$a, b, c$  are existential,  $x, y$  are universal variables

$[a \leftarrow \perp, b \leftarrow \perp, c \leftarrow \top, x \leftarrow \perp, ]$  we are currently at level 5

- $(a_{(2)} \vee b_{(5)} \vee \neg c_{(3)} \vee x_{(1)} \vee y_{(6)})$  is an empty clause
- $(a_{(2)} \vee b_{(5)} \vee c_{(3)} \vee x_{(4)} \vee y_{(1)})$  is **not an empty clause**

# Empty clauses

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

When is a clause an empty clause?

### Empty clause

A clause  $C$  is an empty clause if

- 1 for all  $e \in E(C) : e = \perp$
- 2 for all  $u \in U(C) : u \neq \top$

### Example (Empty clause)

$a, b, c$  are existential,  $x, y$  are universal variables

[ $a \leftarrow \perp, b \leftarrow \perp, c \leftarrow \top, x \leftarrow \perp, y \leftarrow \perp$ ,] we are currently at level 5

- $(a_{(2)} \vee b_{(5)} \vee \neg c_{(3)} \vee x_{(1)} \vee y_{(6)})$  is an empty clause
- $(a_{(2)} \vee b_{(5)} \vee c_{(3)} \vee x_{(4)} \vee y_{(1)})$  is **not an empty clause**

# Further notes

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem

An Algorithm for Q-SAT

Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

- CDCL can be implemented in the same way as in SAT (modified stop criterion)
- Same branching heuristics can be used (wrt. quantification level)
- UP can be done in an similiary way like watched literals

### Analysis in the SAT case

#### Naive backtracking:

- If a variable is assigned, set a flag *flip* to *false*
- If its value is flipped, set *flip* to *true*
- Search the last unflipped universal variable, flip it and backtrack to its level

# Further notes

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

- CDCL can be implemented in the same way as in SAT (modified stop criterion)
- Same branching heuristics can be used (wrt. quantification level)
- UP can be done in an similiary way like watched literals

## Analysis in the SAT case

Naive backtracking:

- If a variable is assigned, set a flag *flip* to *false*
- If its value is flipped, set *flip* to *true*
- Search the last unflipped universal variable, flip it and backtrack to its level

# Planning Problems

Generalized  
SAT Solving

Christoph  
Zengler

## The Bomb in a Toilet Problem

Benchmark	# vars	# clauses	QE	QSAT
toiletA_02_1.2	$2a + 16e$	39	0	0
toiletA_02_1.3	$2a + 24e$	64	0	0
toiletA_02_1.4	$2a + 32e$	89	0.1	0
toiletA_02_5.2	$2a + 48e$	163	0.9	0
toiletA_02_10.2	$2a + 88e$	408	310	0
toiletA_04_1.2	$4a + 28e$	129	0.1	0
toiletA_04_1.3	$4a + 42e$	179	0.4	0
toiletA_04_1.4	$4a + 56e$	229	2	0
toiletA_04_1.5	$4a + 70e$	279	9.7	0
toiletA_04_1.6	$4a + 84e$	329	52.6	0
toiletA_04_1.7	$4a + 98e$	379	436.41	0
toiletA_04_1.8	$4a + 112e$	429	4,120	0
toiletA_04_5.2	$4a + 136e$	894	100.3	0
toiletA_04_10.2	$4a + 16e$	39	59.8	0

Introduction

Propositional Logic

The SAT Problem

Overview

Efficient SAT  
Solving

Unit Propagation

Branching Heuristics

Conflict Driven

Clause Learning

Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem

An Algorithm for  
Q-SAT

Benchmarks

Parametric  
Q-SAT

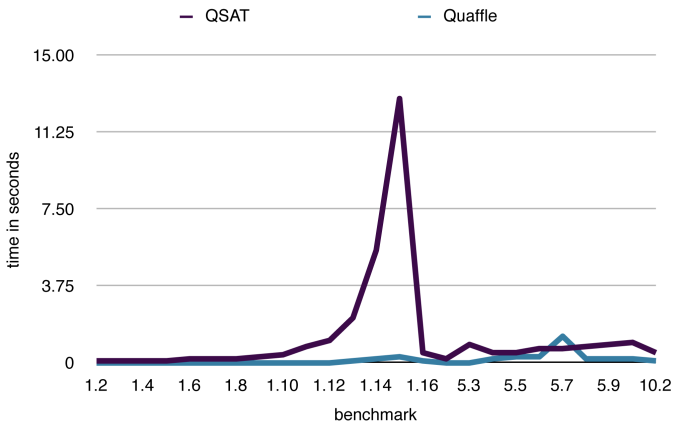
Motivation

The Algorithm

Conclusion

# Comparison to a modern Q-SAT Solver

- Quaffle: Q-SAT Solver from the Boolean Satisfiability Reserach Group at Princeton
- Developed since 2002



Generalized SAT Solving

Christoph Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

Parametric Q-SAT

Motivation  
The Algorithm

Conclusion

# We want more...

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

**Q-SAT:** all formulas over Boolean Algebra in First-Order Logic without free variables.

**We want:** a procedure for *all* formulas over Boolean Algebra in First-Order Logic

### Example

$$\exists x \forall y ((x \vee y \vee w) \wedge (\neg x \vee \neg y \vee z))$$

Output should be:  $z \text{ OR } w$

*This is equivalent to quantifier elimination (QE)!*

# We want more...

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

**Q-SAT:** all formulas over Boolean Algebra in First-Order Logic without free variables.

**We want:** a procedure for *all* formulas over Boolean Algebra in First-Order Logic

### Example

$$\exists x \forall y ((x \vee y \vee w) \wedge (\neg x \vee \neg y \vee z))$$

Output should be:  $z \text{ or } w$

*This is equivalent to quantifier elimination (QE)!*

# We want more...

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

**Q-SAT:** all formulas over Boolean Algebra in First-Order Logic without free variables.

**We want:** a procedure for *all* formulas over Boolean Algebra in First-Order Logic

### Example

$$\exists x \forall y ((x \vee y \vee w) \wedge (\neg x \vee \neg y \vee z))$$

Output should be:  $z \text{ or } w$

*This is equivalent to quantifier elimination (QE)!*

# We want more...

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

**Q-SAT:** all formulas over Boolean Algebra in First-Order Logic without free variables.

**We want:** a procedure for *all* formulas over Boolean Algebra in First-Order Logic

### Example

$$\exists x \forall y ((x \vee y \vee w) \wedge (\neg x \vee \neg y \vee z))$$

Output should be:  $z \text{ or } w$

*This is equivalent to quantifier elimination (QE)!*

# We want more...

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

**Q-SAT:** all formulas over Boolean Algebra in First-Order Logic without free variables.

**We want:** a procedure for *all* formulas over Boolean Algebra in First-Order Logic

### Example

$$\exists x \forall y ((x \vee y \vee w) \wedge (\neg x \vee \neg y \vee z))$$

Output should be:  $z \text{ or } w$

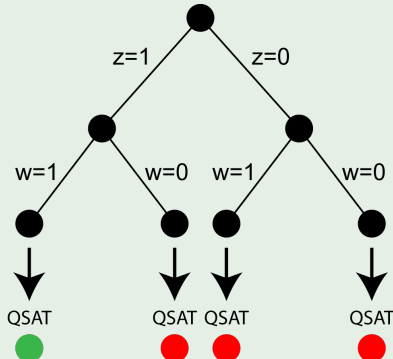
*This is equivalent to quantifier elimination (QE)!*

# The idea

- branch over free variables
- if all free variables are assigned, go into Q-SAT
- branch & bound (early cuts in the branching tree of the free variables)

## Example

$$\forall x \forall y ((x \vee y \vee w) \wedge (\neg x \vee \neg y \vee z))$$



Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# The Parametric Q-SAT Algorithm

- split the set of variables of the formula  $\varphi$  in free and quantified variables:  $\mathcal{F}(\varphi)$  and  $\mathcal{B}(\varphi)$

## the parametric Q-SAT algorithm

PQSAT( $\varphi, \mathcal{F}(\varphi), \alpha$ );

*if all  $x \in \mathcal{F}(\varphi)$  are assigned then*

| Save the result of QSAT( $\varphi$ ) and assignment of  $\alpha$ ;

**end**

$x :=$  choose a free variable of  $\mathcal{F}(\varphi)$ ;

$\alpha' := \alpha \cup [x \leftarrow \top]$ ;

*if no conflict is reached after simplification then*

| PQSAT( $\varphi, \mathcal{F}(\varphi), \alpha'$ );

**end**

$\alpha'' := \alpha \cup [x \leftarrow \perp]$ ;

*if no conflict is reached after simplification then*

| PQSAT( $\varphi, \mathcal{F}(\varphi), \alpha''$ );

**end**

- Same branch and bound mechanism as in DLL

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# The Parametric Q-SAT Algorithm

- split the set of variables of the formula  $\varphi$  in free and quantified variables:  
 $\mathcal{F}(\varphi)$  and  $\mathcal{B}(\varphi)$

## the parametric Q-SAT algorithm

PQSAT( $\varphi, \mathcal{F}(\varphi), \alpha$ );

**if** all  $x \in \mathcal{F}(\varphi)$  are assigned **then**

    | Save the result of QSAT( $\varphi$ ) and assignment of  $\alpha$ ;

**end**

$x :=$  choose a free variable of  $\mathcal{F}(\varphi)$ ;

$\alpha' := \alpha \cup [x \leftarrow \top]$ ;

**if** no conflict is reached after simplification **then**

    | PQSAT( $\varphi, \mathcal{F}(\varphi), \alpha'$ );

**end**

$\alpha'' := \alpha \cup [x \leftarrow \perp]$ ;

**if** no conflict is reached after simplification **then**

    | PQSAT( $\varphi, \mathcal{F}(\varphi), \alpha''$ );

**end**

- Same branch and bound mechanism as in DLL

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# The Parametric Q-SAT Algorithm

- split the set of variables of the formula  $\varphi$  in free and quantified variables:  $\mathcal{F}(\varphi)$  and  $\mathcal{B}(\varphi)$

## the parametric Q-SAT algorithm

PQSAT( $\varphi, \mathcal{F}(\varphi), \alpha$ );

**if** all  $x \in \mathcal{F}(\varphi)$  are assigned **then**

    | Save the result of QSAT( $\varphi$ ) and assignment of  $\alpha$ ;

**end**

$x :=$  choose a free variable of  $\mathcal{F}(\varphi)$ ;

$\alpha' := \alpha \cup [x \leftarrow \top]$ ;

**if** no conflict is reached after simplification **then**

    | PQSAT( $\varphi, \mathcal{F}(\varphi), \alpha'$ );

**end**

$\alpha'' := \alpha \cup [x \leftarrow \perp]$ ;

**if** no conflict is reached after simplification **then**

    | PQSAT( $\varphi, \mathcal{F}(\varphi), \alpha''$ );

**end**

- Same branch and bound mechanism as in DLL

Generalized  
SAT Solving

Christoph  
Zengler

Introduction

Propositional Logic  
The SAT Problem  
Overview

Efficient SAT  
Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven  
Clause Learning  
Benchmarks

The next step:  
Q-SAT

The Q-SAT Problem  
An Algorithm for  
Q-SAT  
Benchmarks

Parametric  
Q-SAT

Motivation  
The Algorithm

Conclusion

# Analysis of the algorithm

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

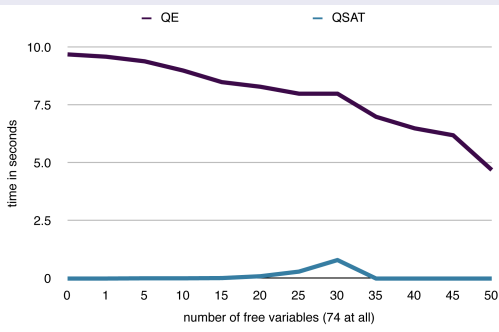
## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

- Theoretical exponential in the number of free variables
- **BUT:** Because of clause learning, after the first runs of QSAT we have a much more compact formula.
- Early Outs are realized with UP: If the UP after an assignment of a free variable leads to a conflict, we cut the branch tree at this node

## Results



# Conclusion

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

### Advantages of the new parametric Q-SAT:

- + a lot more efficient
- + prettier output in DNF
- + enhancements in QSAT are transferred directly to parametric Q-SAT

### Conclusion

Parametric Q-SAT is an efficient alternative for quantifier elimination over Boolean Algebras

### Additional ideas:

- Simplification for output of parametric Q-SAT (iterative consensus)
- Special branching heuristics also for choose of free variables
- Implementation of learning for free variables
- More effective methods for conflict detection than UP

# Conclusion

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

Advantages of the new parametric Q-SAT:

- + a lot more efficient
- + prettier output in DNF
- + enhancements in QSAT are transferred directly to parametric Q-SAT

## Conclusion

Parametric Q-SAT is an efficient alternative for quantifier elimination over Boolean Algebras

Additional ideas:

- Simplification for output of parametric Q-SAT (iterative consensus)
- Special branching heuristics also for choose of free variables
- Implementation of learning for free variables
- More effective methods for conflict detection than UP

# Conclusion

## Generalized SAT Solving

Christoph Zengler

## Introduction

Propositional Logic  
The SAT Problem  
Overview

## Efficient SAT Solving

Unit Propagation  
Branching Heuristics  
Conflict Driven Clause Learning  
Benchmarks

## The next step: Q-SAT

The Q-SAT Problem  
An Algorithm for Q-SAT  
Benchmarks

## Parametric Q-SAT

Motivation  
The Algorithm

## Conclusion

Advantages of the new parametric Q-SAT:

- + a lot more efficient
- + prettier output in DNF
- + enhancements in QSAT are transferred directly to parametric Q-SAT

## Conclusion

Parametric Q-SAT is an efficient alternative for quantifier elimination over Boolean Algebras

## Additional ideas:

- Simplification for output of parametric Q-SAT (iterative consensus)
- Special branching heuristics also for choose of free variables
- Implementation of learning for free variables
- More effective methods for conflict detection than UP