

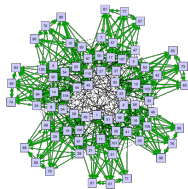
# Parametric Quantified SAT Solving

Thomas Sturm<sup>1</sup>   Christoph Zengler<sup>2</sup>

<sup>1</sup>Departamento de Matemáticas, Estadística y Computación  
Universidad de Cantabria, Santander, Spain

<sup>2</sup>Symbolic Computation Group  
Wilhelm-Schickard-Institute of Informatics  
Universität Tübingen, Germany

26-07-2010



# Motivation

## Example (QE for quantified propositional formulas)

**Given:** Formula  $\varphi$  with propositional variables  $x, y, u, w$ :

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

**Wanted:** an equivalent formula  $\varphi'$  without quantifiers.

**One possible solution:**  $\varphi' = (u \wedge w) \vee (\neg u \wedge w)$

[Seidl&Sturm,2003] Embedding of propositional logic into first-order logic:

- Use language  $L = (0, 1, !, \&, |)$  of Boolean algebras
- Replace  $X_i$  and  $\neg X_i$  with  $x_i = 1$  and  $x_i = 0$ , resp.
- Implementation of a QE based on *substitute-and-simplify* in Redlog

## Our idea

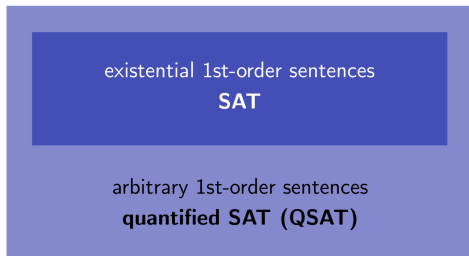
- **Observation:** Despite equal theoretical complexity, current implementations of SAT algorithms perform better on real-world applications than SS-QE
- **Idea:** Use SAT algorithms for QE

existential 1st-order sentences  
SAT

i.e.  $\exists x \exists y (x \vee y) \wedge (\neg x \vee \neg y)$

## Our idea

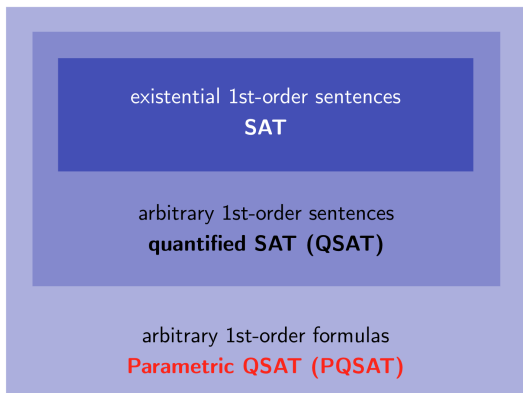
- **Observation:** Despite equal theoretical complexity, current implementations of SAT algorithms perform better on real-world applications than SS-QE
- **Idea:** Use SAT algorithms for QE



$$\text{i.e. } \forall x \exists y (x \vee y) \wedge (\neg x \vee \neg y)$$

## Our idea

- **Observation:** Despite equal theoretical complexity, current implementations of SAT algorithms perform better on real-world applications than SS-QE
- **Idea:** Use SAT algorithms for QE



$$\text{i.e. } \exists x \forall y (x \vee y) \wedge (\neg x \vee \neg y \vee w) \wedge \neg z$$

# The Story of SAT Solving

## DPLL Algorithm

[Davis&Putnam,1960] and [Davis&Logemann&Loveland,1962]

**Idea:** Complete search over  $2^n$  variable assignments with early cuts for unsatisfiable branches

- Restriction to CNF
- Fast unit propagation
- Elaborate heuristics for variable selection

# The Story of SAT Solving

## DPLL Algorithm

[Davis&Putnam,1960] and [Davis&Logemann&Loveland,1962]

**Idea:** Complete search over  $2^n$  variable assignments with early cuts for unsatisfiable branches

- Restriction to CNF
- Fast unit propagation
- Elaborate heuristics for variable selection

## The Quantum Leap: Clause Learning

[Marques-Silva&Sakallah,1996]

Avoid repetitive bad assignments

- Deduce new information when a conflict is detected (resolution)
- Learn this new information to avoid that particular mistake in future

# A SAT Example

$\{\neg u, w\}, \{\neg u, \neg m, h\}, \{\neg u, \neg f, m\}, \{\neg f, \neg g, \neg h\}, \{y, f\}, \{\neg f, g\}$

Decisions

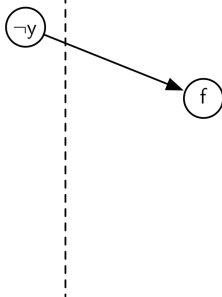
$\neg y$

Variable	Value	Reason
y	<b>F</b>	decision

# A SAT Example

$$\{\neg u, w\}, \{\neg u, \neg m, h\}, \{\neg u, \neg f, m\}, \{\neg f, \neg g, \neg h\}, \{y, f\}, \{\neg f, g\}$$

Decisions

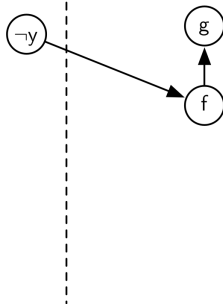


Variable	Value	Reason
y	<b>F</b>	decision
f	<b>T</b>	{y,f}

# A SAT Example

$$\{\neg u, w\}, \{\neg u, \neg m, h\}, \{\neg u, \neg f, m\}, \{\neg f, \neg g, \neg h\}, \{y, f\}, \{\neg f, g\}$$

Decisions

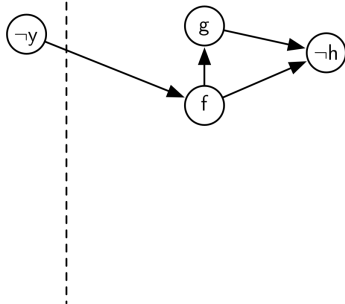


Variable	Value	Reason
y	<b>F</b>	decision
f	<b>T</b>	{y,f}
g	<b>T</b>	{¬f,g}

# A SAT Example

$\{\neg u, w\}, \{\neg u, \neg m, h\}, \{\neg u, \neg f, m\}, \{\neg f, \neg g, \neg h\}, \{y, f\}, \{\neg f, g\}$

Decisions

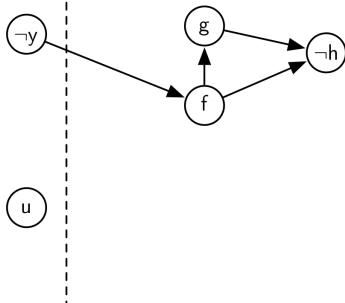


Variable	Value	Reason
y	<b>F</b>	decision
f	<b>T</b>	$\{y, f\}$
g	<b>T</b>	$\{\neg f, g\}$
h	<b>F</b>	$\{\neg f, \neg g, \neg h\}$

# A SAT Example

$$\{\neg u, w\}, \{\neg u, \neg m, h\}, \{\neg u, \neg f, m\}, \{\neg f, \neg g, \neg h\}, \{y, f\}, \{\neg f, g\}$$

Decisions

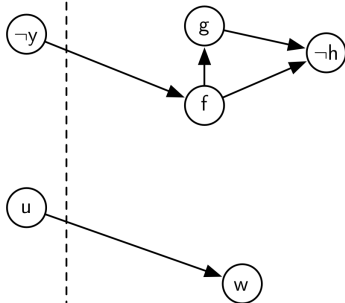


Variable	Value	Reason
y	<b>F</b>	decision
f	<b>T</b>	{y,f}
g	<b>T</b>	{¬f,g}
h	<b>F</b>	{¬f,¬g,¬h}
u	<b>T</b>	decision

# A SAT Example

$\{\neg u, w\}$ ,  $\{\neg u, \neg m, h\}$ ,  $\{\neg u, \neg f, m\}$ ,  $\{\neg f, \neg g, \neg h\}$ ,  $\{y, f\}$ ,  $\{\neg f, g\}$

Decisions

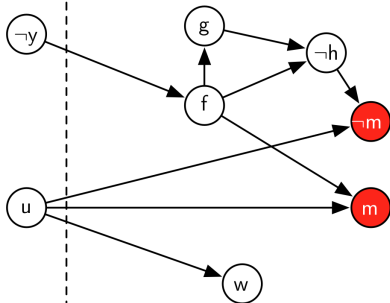


Variable	Value	Reason
y	<b>F</b>	decision
f	<b>T</b>	$\{y, f\}$
g	<b>T</b>	$\{\neg f, g\}$
h	<b>F</b>	$\{\neg f, \neg g, \neg h\}$
u	<b>T</b>	decision
w	<b>T</b>	$\{\neg u, w\}$

# A SAT Example

$\{\neg u, w\}, \{\neg u, \neg m, h\}, \{\neg u, \neg f, m\}, \{\neg f, \neg g, \neg h\}, \{y, f\}, \{\neg f, g\}$

Decisions

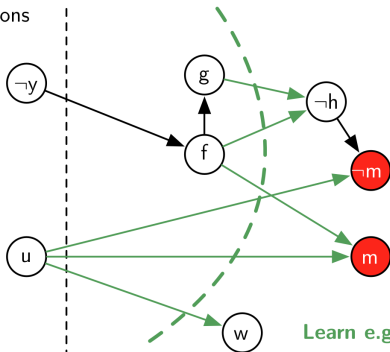


Variable	Value	Reason
y	<b>F</b>	decision
f	<b>T</b>	{y,f}
g	<b>T</b>	{¬f,g}
h	<b>F</b>	{¬f,¬g,¬h}
u	<b>T</b>	decision
w	<b>T</b>	{¬u,w}
m	<b>F</b>	{¬u,¬m,h}
m	<b>T</b>	{¬u,¬f,m}

# A SAT Example

$\{\neg u, w\}, \{\neg u, \neg m, h\}, \{\neg u, \neg f, m\}, \{\neg f, \neg g, \neg h\}, \{y, f\}, \{\neg f, g\}$

Decisions



Variable	Value	Reason
y	<b>F</b>	decision
f	<b>T</b>	$\{y, f\}$
g	<b>T</b>	$\{\neg f, g\}$
h	<b>F</b>	$\{\neg f, \neg g, \neg h\}$
u	<b>T</b>	decision
w	<b>T</b>	$\{\neg u, w\}$
m	<b>F</b>	$\{\neg u, \neg m, h\}$
m	<b>T</b>	$\{\neg u, \neg f, m\}$

Learn e.g.:  $\neg(g \wedge f \wedge u) = (\neg g \vee \neg f \vee \neg u)$

# Where are we now?

existential 1st-order sentences  
**SAT**



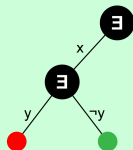
arbitrary 1st-order sentences  
**quantified SAT (QSAT)**

arbitrary 1st-order formulas  
**Parametric QSAT (PQSAT)**

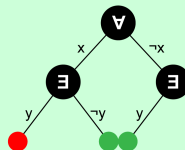
# QSAT solving (aka QBF)

- Backtracking for both, unsatisfying branches and satisfying branches of universally quantified variables

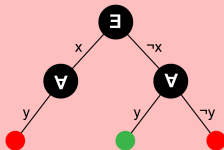
$$\exists x \exists y ((x \vee y) \wedge (\neg x \vee \neg y))$$



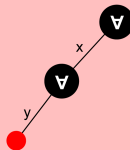
$$\forall x \exists y ((x \vee y) \wedge (\neg x \vee \neg y))$$



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



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



# Where are we now?

existential 1st-order sentences  
**SAT** ✓

arbitrary 1st-order sentences  
**quantified SAT (QSAT)** ✓

arbitrary 1st-order formulas  
**Parametric QSAT (PQSAT)**

# PQSAT Solving

## PQSAT

- **Input:** propositional formula  $\varphi$  with arbitrary quantification (also free variables)
- **Output:** DNF, establishing necessary and sufficient conditions on the free variables for the existence of a satisfying assignment

# PQSAT Solving

## PQSAT

- **Input:** propositional formula  $\varphi$  with arbitrary quantification (also free variables)
- **Output:** DNF, establishing necessary and sufficient conditions on the free variables for the existence of a satisfying assignment

### Idea

- DPLL-style top algorithm with QSAT as black-box decision procedure

$\Gamma = \emptyset$

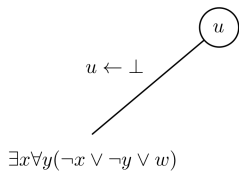
**foreach** complete assignment  $\alpha$  of the free variables **do**

**if** QSAT( $\varphi, \alpha$ ) yields  $\tau = \text{true}$  **then**  
     └  $\Gamma = \Gamma \cup \{\alpha\}$

**return** DNF representation of all  $\alpha \in \Gamma$

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

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



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

$u$

$$u \leftarrow \perp$$

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

$w$

$$w \leftarrow \perp$$

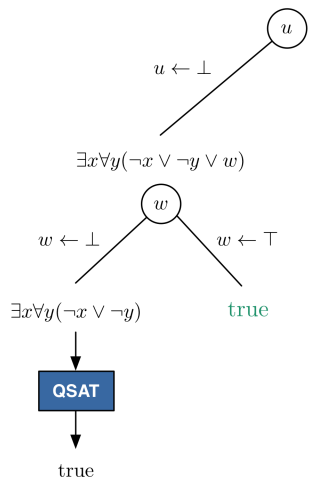
$$\exists x \forall y (\neg x \vee \neg y)$$

QSAT

true

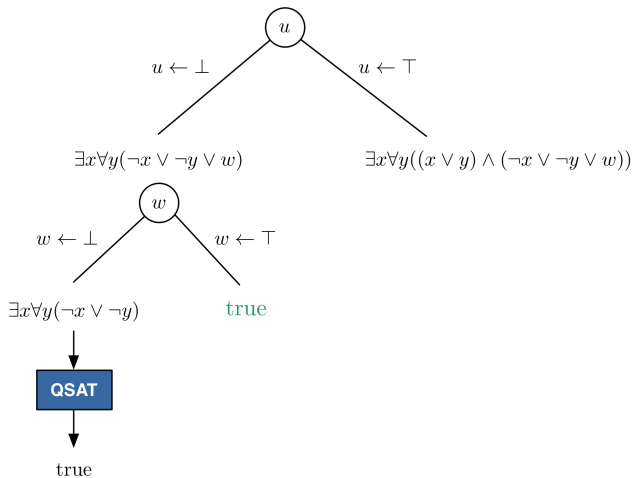
keep  $(\neg u \wedge \neg w)$

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

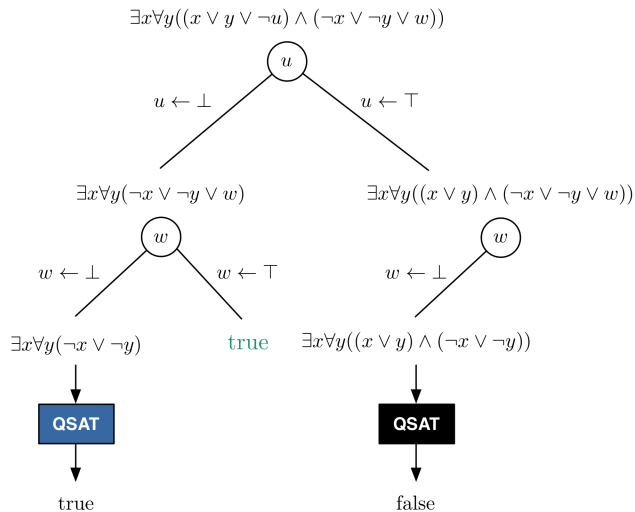


keep  $(\neg u \wedge \neg w)$     keep  $(\neg u \wedge w)$

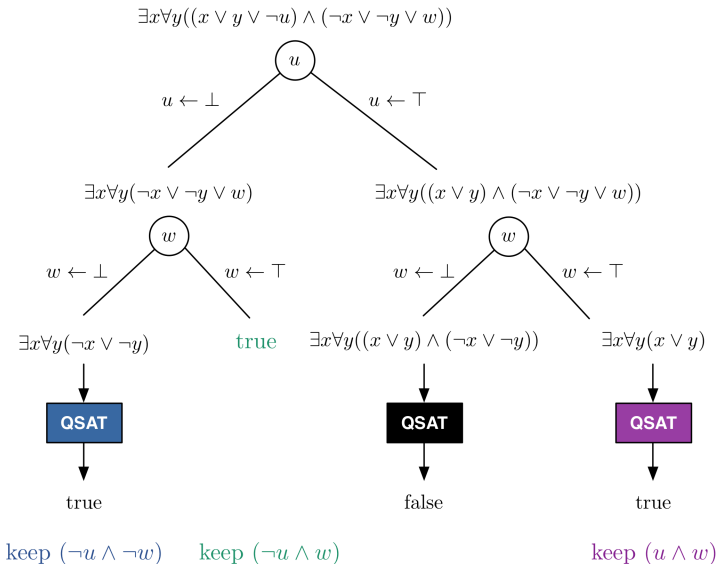
$$\exists x \forall y ((x \vee y \vee \neg u) \wedge (\neg x \vee \neg y \vee w))$$

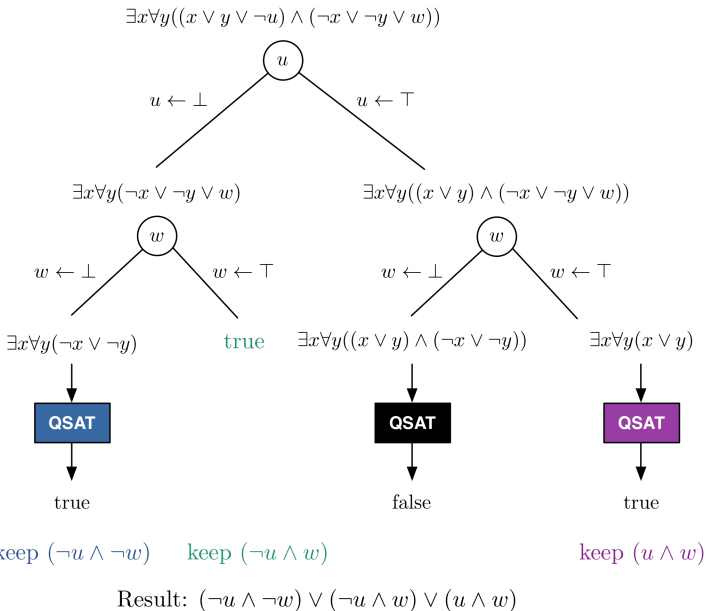


keep  $(\neg u \wedge \neg w)$       keep  $(\neg u \wedge w)$



keep  $(\neg u \wedge \neg w)$       keep  $(\neg u \wedge w)$

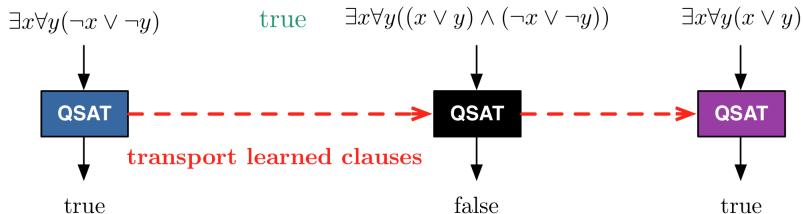




# PQSAT: Details

Use all improvements of CDCL for PQSAT:

- Unit propagation after assigning free variables  
 ⇒ early detection of implication and unsatisfiable branches
- Transport learned clauses from one QSAT call to the next
- Activity heuristics: Reduce the size of formula by deleting unused learned clauses after each QSAT run



# PQSAT Properties

## Correctness

- $\varphi$  a quantified propositional formula
- $(\tau, \varphi') = \text{PQSAT}(\varphi, \emptyset)$

$\tau$  is quantifier-free and  $\tau \longleftrightarrow \varphi$ .

## Termination

PQSAT terminates.

## Complexity

- $f = |\text{fvars}(\varphi)|$
- $b = |\text{bvars}(\varphi)|$

**Asymptotic time complexity** bounded by  $2^{f+b}$  in the worst case  
**In particular** bounded by  $2^{\text{length}(\varphi)}$ .

# Example: Automobile Configuration

[Küchlin&Sinz,2000]

- Cars are customized products
- Each part  $p$  is mapped to an equipment code  $x_p$
- Parts can be *customer options* or *hidden parts*
- Set of all constructible cars is described by product overview formula (POF)

## POF

Conjunction of two different rule types:

- Constructibility conditions:  $x_p \rightarrow \varphi$
- Supplementary codes:  $\varphi \rightarrow x_p$

with  $\varphi$  quantifier-free propositional formula

## Example (POF)

Customer options:

- Engines:  $e_1, e_2, e_3$
- Air conditioning:  $a_1, a_2, a_3$

Hidden Parts:

- Gearboxes:  $g_1, g_2$

## Example (POF)

Customer options:

- Engines:  $e_1, e_2, e_3$
- Air conditioning:  $a_1, a_2, a_3$

Hidden Parts:

- Gearboxes:  $g_1, g_2$

**Exactly one engine**

$$cc_1 = \text{true} \rightarrow e_1 \vee e_2 \vee e_3,$$

$$cc_2 = e_1 \rightarrow \neg e_2 \wedge \neg e_3,$$

$$cc_3 = e_2 \rightarrow \neg e_1 \wedge \neg e_3,$$

$$cc_4 = e_3 \rightarrow \neg e_1 \wedge \neg e_2.$$

**Exactly one gearbox**

$$cc_5 = \text{true} \rightarrow g_1 \vee g_2,$$

$$cc_6 = g_1 \rightarrow \neg g_2,$$

$$cc_7 = g_2 \rightarrow \neg g_1.$$

## Example (POF)

Customer options:

- Engines:  $e_1, e_2, e_3$
- Air conditioning:  $a_1, a_2, a_3$

**Exactly one engine**

$$CC_1 = \text{true} \rightarrow e_1 \vee e_2 \vee e_3,$$

$$CC_2 = e_1 \rightarrow \neg e_2 \wedge \neg e_3,$$

$$CC_3 = e_2 \rightarrow \neg e_1 \wedge \neg e_3,$$

$$CC_4 = e_3 \rightarrow \neg e_1 \wedge \neg e_2.$$

**Exactly one gearbox**

$$CC_5 = \text{true} \rightarrow g_1 \vee g_2,$$

$$CC_6 = g_1 \rightarrow \neg g_2,$$

$$CC_7 = g_2 \rightarrow \neg g_1.$$

Hidden Parts:

- Gearboxes:  $g_1, g_2$

**Relations between engines and gearboxes**

$$CC_8 = e_1 \rightarrow g_1,$$

$$CC_9 = e_2 \rightarrow g_2,$$

$$CC_{10} = e_3 \rightarrow g_1 \vee g_2.$$

**Restrictions for the air conditioning**

$$CC_{11} = a_2 \rightarrow \neg a_3,$$

$$SC_1 = e_3 \wedge g_1 \rightarrow a_2,$$

$$SC_2 = e_3 \wedge g_2 \rightarrow a_3.$$

## Example (POF)

Customer options:

- Engines:  $e_1, e_2, e_3$
- Air conditioning:  $a_1, a_2, a_3$

**Exactly one engine**

$$cc_1 = \text{true} \rightarrow e_1 \vee e_2 \vee e_3,$$

$$cc_2 = e_1 \rightarrow \neg e_2 \wedge \neg e_3,$$

$$cc_3 = e_2 \rightarrow \neg e_1 \wedge \neg e_3,$$

$$cc_4 = e_3 \rightarrow \neg e_1 \wedge \neg e_2.$$

**Exactly one gearbox**

$$cc_5 = \text{true} \rightarrow g_1 \vee g_2,$$

$$cc_6 = g_1 \rightarrow \neg g_2,$$

$$cc_7 = g_2 \rightarrow \neg g_1.$$

Hidden Parts:

- Gearboxes:  $g_1, g_2$

**Relations between engines and gearboxes**

$$cc_8 = e_1 \rightarrow g_1,$$

$$cc_9 = e_2 \rightarrow g_2,$$

$$cc_{10} = e_3 \rightarrow g_1 \vee g_2.$$

**Restrictions for the air conditioning**

$$cc_{11} = a_2 \rightarrow \neg a_3,$$

$$sc_1 = e_3 \wedge g_1 \rightarrow a_2,$$

$$sc_2 = e_3 \wedge g_2 \rightarrow a_3.$$

$$\text{POF} = \bigwedge_{i=1}^{11} cc_i \wedge \bigwedge_{j=1}^2 sc_j$$

# The Order Process

## Order Process

- 1 Set all codes  $x_p$  of chosen parts  $p$  to  $\top$ , set all other codes to  $\perp$
- 2 Use supplementary rules to complete order  
 $\Rightarrow$  swap codes from  $\perp$  to  $\top$
- 3 Check constructibility conditions

# The Order Process

## Order Process

- 1 Set all codes  $x_p$  of chosen parts  $p$  to  $\top$ , set all other codes to  $\perp$
- 2 Use supplementary rules to complete order  
 $\Rightarrow$  swap codes from  $\perp$  to  $\top$
- 3 Check constructibility conditions

## Step 2 ...

- ... yields the final configuration of the car
- ... is bounded by certain rules and is strongly dependent on the order of adding assignments
- ... can render the POF unsatisfiable although the vehicle is constructible

# The Order Process

## Order Process

- 1 Set all codes  $x_p$  of chosen parts  $p$  to  $\top$ , set all other codes to  $\perp$
- 2 Use supplementary rules to complete order  
 $\Rightarrow$  swap codes from  $\perp$  to  $\top$
- 3 Check constructibility conditions

## Step 2 ...

- ... yields the final configuration of the car
- ... is bounded by certain rules and is strongly dependent on the order of adding assignments
- ... can render the POF unsatisfiable although the vehicle is constructible

**Solution:** Use PQSAT to automatically generate repair sets

# Generating Repair Sets with PQSAT

## Our approach

- 1 Use initial customer order and add  $\wedge x_p$  to POF for each chosen part  $p$
- 2  $\exists$ -quantify all codes corresponding to hidden parts
- 3 Compute  $\bigvee_i \tau_i$  with PQSAT
  - each  $\tau_i$  is one possible way of rendering the vehicle constructible

# Generating Repair Sets with PQSAT

## Our approach

- 1 Use initial customer order and add  $\wedge x_p$  to POF for each chosen part  $p$
- 2  $\exists$ -quantify all codes corresponding to hidden parts
- 3 Compute  $\bigvee_i \tau_i$  with PQSAT
  - each  $\tau_i$  is one possible way of rendering the vehicle constructible

## Example (Generating Repair Sets)

Customer chose  $e_3$  and  $a_1$  and order procedure returned false

- 1 Create new POF:  $\text{POF}' = \bigwedge_{i=1}^{11} cc_i \wedge \bigwedge_{j=1}^2 sc_j \wedge e_3 \wedge a_1$
- 2 Quantify hidden parts:  $\exists g_1 \exists g_2 \text{POF}'$
- 3 Compute PQSAT  $(\exists g_1 \exists g_2 \text{POF}', \emptyset) = \tau_1 \vee \tau_2$  with  
 $\tau_1 = \neg e_2 \wedge \neg e_1 \wedge a_3 \wedge \neg a_2$  and  $\tau_2 = \neg e_2 \wedge \neg e_1 \wedge \neg a_3 \wedge a_2$ .

# Benchmarks

- SAT, QSAT, and PQSAT are implemented in the Redlog package of Reduce<sup>1</sup>
- We can handle larger problems than the original SS-QE
- Speedup between 10 and 300 compared to SS-QE
- For problems with a large number of free quantifiers SS-QE and PQSAT perform similarly
- Real industrial problems (e.g. Daimler benchmarks) are feasible
- **But:** SAT and QSAT cannot compete with specialized implementations like MiniSat, PicoSat, Quantor,...

---

<sup>1</sup><http://reduce-algebra.sourceforge.net>

# Conclusion & Outlook

## Conclusion

- Generalization of successful algorithmic ideas from SAT and QSAT to the parametric case
- Efficient implementation in Redlog
- PQSAT can be easily implemented with other QSAT solvers as black-box
- Application on real industrial problems

# Conclusion & Outlook

## Conclusion

- Generalization of successful algorithmic ideas from SAT and QSAT to the parametric case
- Efficient implementation in Redlog
- PQSAT can be easily implemented with other QSAT solvers as black-box
- Application on real industrial problems

## Future Work

- Adopt more ideas from DPLL
  - Variable heuristics
  - Learning for free variables
- Non-CNF version based on non-CNF SAT and QSAT solving
- Applications in bounded model checking and circuit optimization