

# Quantifier Elimination with SAT Solving Algorithms

Christoph Zengler

Talk at the University of Tübingen  
Symbolic Computation Group  
Prof. Küchlin

04.12.2008

## Example

**Consider:** a formula  $\phi$ :

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

**Question:** an equivalent formula  $\phi'$  without quantifiers.

**Solution:**  $(u \wedge w) \vee (\neg u \wedge w)$

### Plan

- 1 Quantifier Elimination for fully existential quantified formulas  
*with SAT Solving*
- 2 Quantifier Elimination for fully quantified formulas  
*with Q-SAT Solving*
- 3 Quantifier Elimination for arbitrarily First-Order Logic formulas  
*with parametric Q-SAT Solving*

Formulas with only existential variables (no free variables)

# SAT

Formulas with only existential variables (no free variables)

SAT

Formulas with existential and universal variables (no free variables)

Q-SAT

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

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

## Definition (Language)

A language  $\mathcal{L}$  fixes sets for function symbols  $f$  and relation symbols  $R$  with their corresponding arity  $\sigma(R)$  and  $\sigma(f)$  respectively.

We consider the language

$$\mathcal{L} = (1^{(0)}, 0^{(0)}, !^{(1)}, \&^{(2)}, |^{(2)})$$

## Definition (Boolean Algebra)

Structure (Algebra)  $\mathbb{B} = (\{1, 0\}, \mathcal{L})$  with 2-element universe  $\{1, 0\}$  and interpretations:

! as *not*

& as *and*

| as *or*

## Definition (Terms)

- Variables of a fixed, infinite set of variables  $\mathcal{V}$
- constants 1, 0
- $!t$  for a term  $t$
- $t_1 \& t_2$  or  $t_1 | t_2$  for terms  $t_1, t_2$

## Definition (Atomic formulas)

- Equations  $t_1 = t_2$  for terms  $t_1, t_2$

## Definition (First-Order Logic formulas)

- Connection of atomic formulas with Boolean operators  $\neg, \wedge$  and  $\vee$
- Quantification of variables with  $\exists x$  or  $\forall x$

## Example

- **Terms:**  $1, 0, x, y, z, \dots, x|y, !z$
- **Atomic formulas:**  $x = 1, z = !(y \& u)$
- **First-Order Logic formulas:**  $x = 1 \wedge z = !(y \& u), \forall x \exists y (x = 1 \vee y = 0)$

## Definition (Quantifier Elimination)

An algorithm which computes for each First-Order Logic formula  $\phi$  an equivalent quantifier free formula  $\phi'$  with  $\phi \iff \phi'$  is referred to as quantifier elimination procedure (QE).

## Example

$\phi$	$\phi'$
$x \& y$	$x \& y$
$\exists x(x = 1)$	<i>true</i>
$\forall x(x = 1)$	<i>false</i>
$\exists x \forall y ((x = 1 \vee y = 1 \vee u = 0))$	
$\wedge (x = 0 \vee y = 0 \vee w = 1) \wedge (u = 1 \vee w = 1)$	$(u = 1 \wedge w = 1) \vee (u = 1 \wedge w = 1)$

All presented algorithms and procedures are implemented in **REDLOG**, a module for First-Order Logic for the computer algebra system REDUCE.

- Reads .dimacs and .qdimacs files
- SAT Solving for Propositional Logic (via a propositional wrapper)
- Q-SAT Solving and parametric Q-SAT Solving for quantified problems

More QE procedures for ...

- ... ordered fields (e.g. real numbers)
- ... algebraically closed fields (e.g. complex numbers)
- ... Queues
- ... (uniform) Pressburger Arithmetic

Propositional Logic in our framework:

- Instead of propositional variables – atomic formulas
- $X$  is translated  $x = 1$ ,  $\neg X$  is translated  $x = 0$

## Example

The following formula in Propositional Logic

$$(A \vee B \vee C) \wedge (A \vee \neg B) \wedge (B \vee \neg C)$$

is translated in the following formula in First-Order Logic:

$$(a = 1 \vee b = 1 \vee c = 1) \wedge (a = 1 \vee b = 0) \wedge (b = 1 \vee c = 0)$$

Thus each unquantified First-Order Logic formula can be translated in a corresponding formula in Propositional Logic and vice versa.

We can use the same translation for a fully existential quantified (no free variables) formula in First-Order Logic.

⇒ **Reduction to a SAT problem**

Implementation of a SAT-Solvers with State-of-the-Art techniques:

- Preprocessing of the input formula
- Different selection heuristics (DLCS, DLIS, MOM, ZMOM, Activity)
- Conflict Driven Clause Learning
- Unit Propagation with Watched Literals
- Deletion of learned clauses with low activity
- Restart after a certain number of learned clauses
- Completely configurable

## Speed

Beats the existing QE for Boolean Algebras by far but cannot compete with modern SAT Solvers.

- Only lists were used for implementation
- No efficient data structures in R-LISP
- No influence at the heap organization

⇒ Problems with thousands of variables and tens of thousands of clauses are now feasible.

## Demo time...

- Verification of a small DLX processor with 6 instruction types

We already know:

## Example

The SAT problem

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

can be expressed as first order formula:

$$\exists a \exists b \exists c ((a = 1 \vee b = 1 \vee c = 1) \wedge (a = 1 \vee b = 0) \wedge (b = 0 \vee c = 0))$$

**And now?:** Extension for existential **and universal** variables.

## Q-SAT

The quantified SAT problem (Q-SAT) is the question whether a given fully quantified formula is true or false.

### Example (Q-SAT problems)

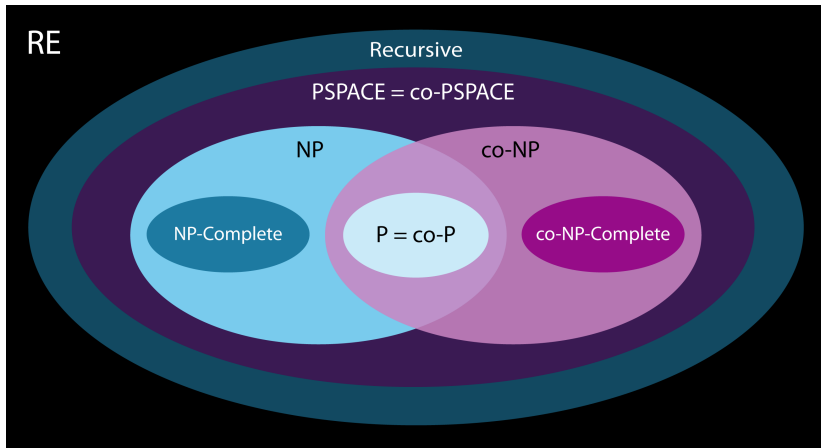
$$\forall x \exists y ((x \vee y) \wedge (\neg x \vee \neg y)) = \text{true}$$

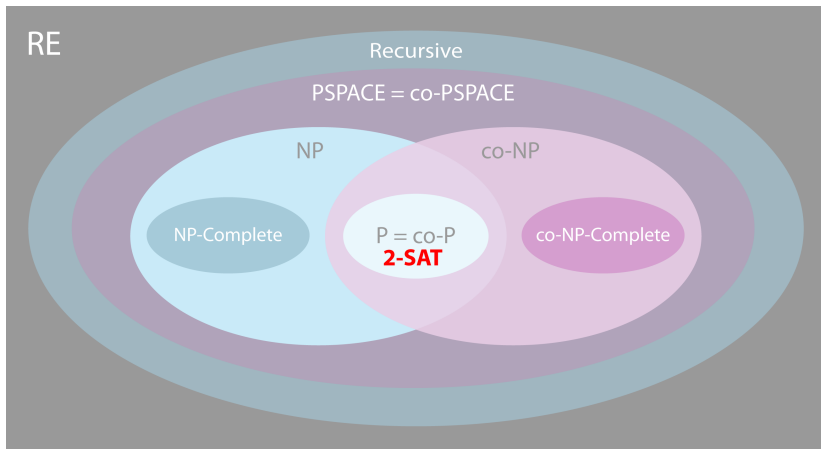
$$\forall x \forall y ((x \vee y) \wedge (\neg x \vee \neg y)) = \text{false}$$

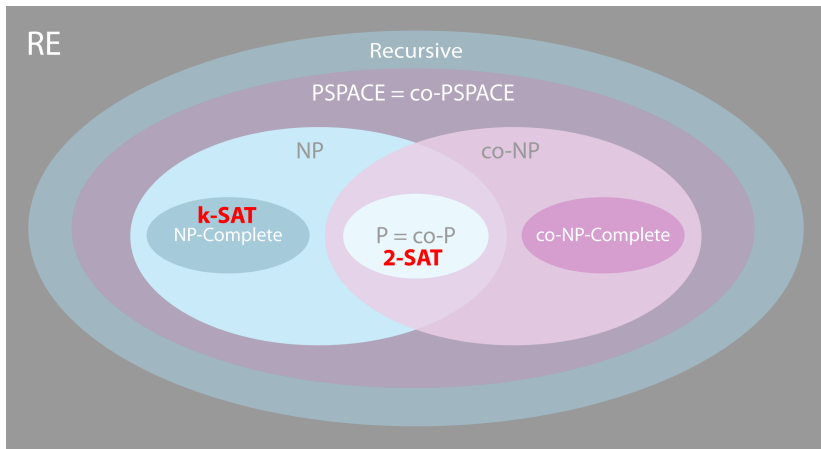
- Formula must be in prenex normal form (PNF)
- Group quantifiers of same type at one level (quantifier level  $q_l$ )
- Heuristics must obey quantification level

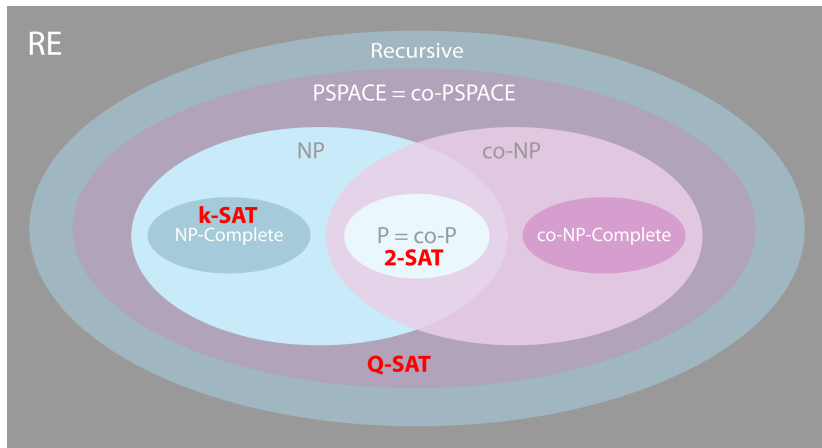
### Example

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



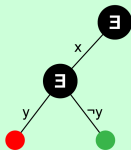




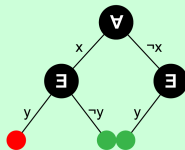


- 2 different kinds of node: existential and universal nodes
- existential nodes require 1 satisfying branch, universal nodes require 2 satisfying branches

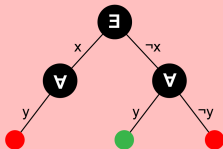
$$\exists x \exists y ((x \vee y) \wedge (\neg x \vee \neg y))$$



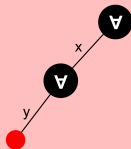
$$\forall x \exists y ((x \vee y) \wedge (\neg x \vee \neg y))$$



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



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



- Backtracking also in the satisfying case

## SAT vs. Q-SAT Algorithm

### SAT Algorithm

```

level := 0;
while true do
  UnitPropgation(f,α);
  if a conflict is reached then
    level := analyseConflict();
    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
end
  
```

### Q-SAT Algorithm

```

level := 0;
while true do
  UnitPropgation(f,α);
  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;
      choose an unassigned  $x \in \mathcal{V}(F)$ 
      (wrt. the q-level);
       $\alpha := \alpha \cup [x \leftarrow 0]$ ;
    end
  end
end
end
  
```

**Rule for SAT:** A clause is empty if is not satisfied yet but all literals are set.

**New rule for Q-SAT:**

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

## Empty clause

A clause  $C$  is an empty clause if

- 1 for all  $e \in E(C) : e = 0$
- 2 for all  $u \in U(C) : u \neq 1$

## Example (Empty clause)

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

[ $a \leftarrow 1, b \leftarrow 0, c \leftarrow 1, x \leftarrow 0,$ ]

- $(\neg a \vee b \vee \neg c \vee x \vee y)$  is an empty clause

**Rule for SAT:** A clause is unit, if it is not satisfied yet and only one literal is not yet set.

**New Rule for Q-SAT:**

## Unit clauses

A clause  $C$  is unit if

- 1 it exists  $e \in E(C)$ ,  $e = nil$ .
- 2 For any  $e' \in E(C)$ ,  $e' \neq e$ :  $e' = 0$
- 3 for all  $u \in U(C)$  with  $u \neq 1$ :  $u = nil \Rightarrow ql(u) > ql(e)$

## Example (Unit Clause)

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

$[a \leftarrow 0, c \leftarrow 1, x \leftarrow 0]$ , the next implied variable  $b$  would have  $ql = 5$ .

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

### Further notes

- 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 a similar way like watched literals

## Analysis in the SAT case

### Naive backtracking:

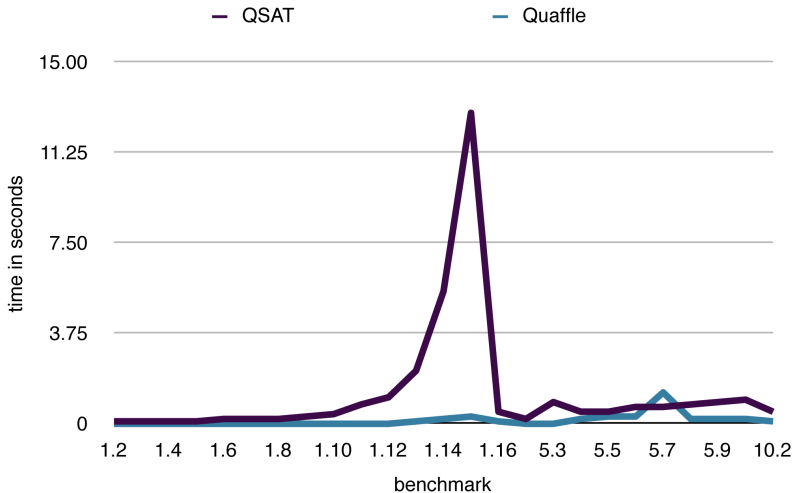
- If a variable is assigned, set a flag *flip* to *false*
- If its value is flipped, set *flip* to true
- Search the last unflipped universal variable, flip it and backtrack to its level

## 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

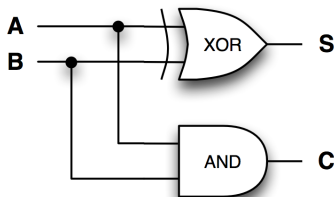
## Comparison to a modern Q-SAT Solver

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



## Demo time...

- A standard Q-SAT Benchmark Planning problem: Bomb in the toilet
- A half adder

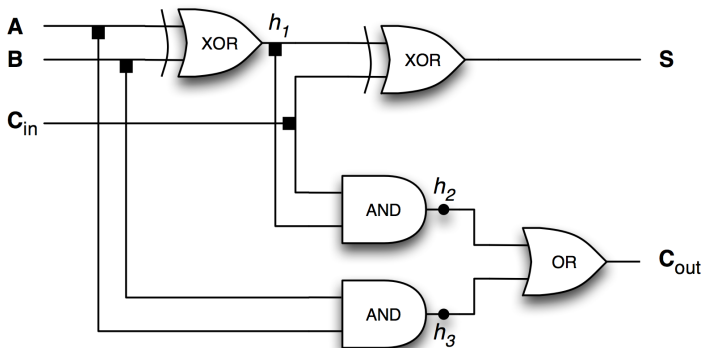


$$ha := s = xor(a, b) \text{ and } c = (a \& b)$$

**Question 1:** Is there an input  $(a, b)$  for every possible output  $(s, c)$ :

$$\forall s \forall c \exists a \exists b (ha)$$

- A full adder



$$fa := \exists h_1 \exists h_2 \exists h_3 (h_1 = xor(a, b) \wedge s = xor(c_{in}, h_1) \wedge h_2 = (h_1 \& c_{in}) \wedge h_3 = (a \& b) \wedge c_{out} = (h_2 | h_3))$$

**Question 2:** Is there an input  $(a, b, c_{in})$  for every possible output  $(s, c_{out})$ :

$$\forall s \forall c_{out} \exists a \exists b \exists c_{in} (fa)$$

**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:  $z \text{ OR } w$

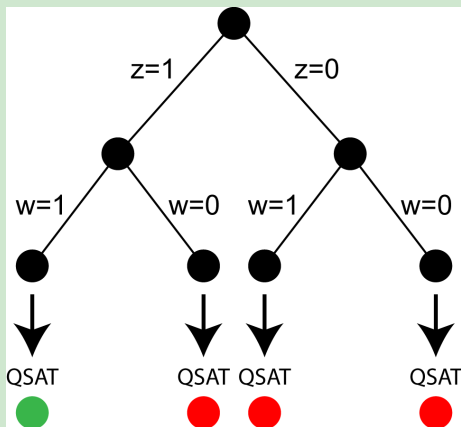
*This is equivalent to quantifier elimination (QE)!*

**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**

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



**The 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**

```

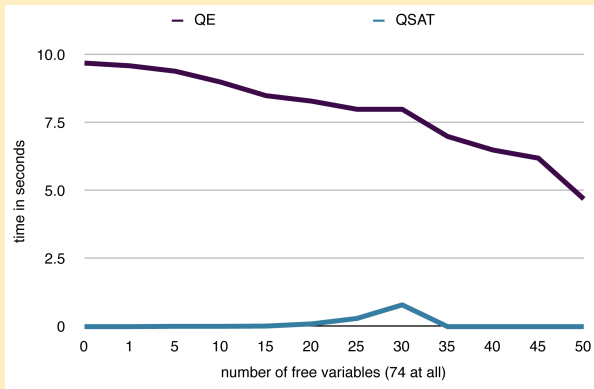
PQSAT( $\varphi, \mathcal{F}(\varphi), \alpha$ );
if all  $x \in \mathcal{F}(\varphi)$  are assigned then
  | Save the result of QSAT( $\varphi$ ) and assignment of  $\alpha$ ;
end
 $x :=$  choose a free variable of  $\mathcal{F}(\varphi)$ ;
 $\alpha' := \alpha \cup [x \leftarrow 1]$ ;
if no conflict is reached after simplification then
  | PQSAT( $\varphi, \mathcal{F}(\varphi), \alpha'$ );
end
 $\alpha'' := \alpha \cup [x \leftarrow 0]$ ;
if no conflict is reached after simplification then
  | PQSAT( $\varphi, \mathcal{F}(\varphi), \alpha''$ );
end

```

- Same branch and bound mechanism as in DLL

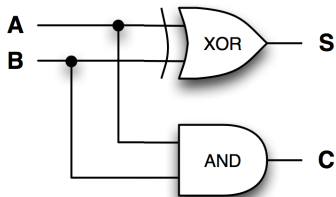
- Theoretical exponential in the number of free variables
- **BUT:** Because of clause learning, after the first runs of QSAT we have a an enhanced formula.
- Early Outs are realized with UP: If the UP after an assignment of a free variable leads to a conflict, we cut the branch tree at this node

## Results



## Demo time...

- Remember the half adder

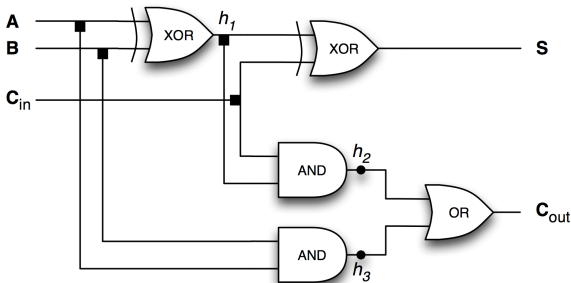


$$ha := s = xor(a, b) \text{ and } c = (a \& b)$$

**Question 3:** Which outputs exist? (codomain of the circuit):

$$\exists a \exists b (ha)$$

- Remember the full adder



$$fa := \exists h_1 \exists h_2 \exists h_3 (h_1 = xor(a, b) \wedge s = xor(c_{in}, h_1) \wedge h_2 = (h_1 \& c_{in}) \wedge h_3 = (a \& b) \wedge c_{out} = (h_2 | h_3))$$

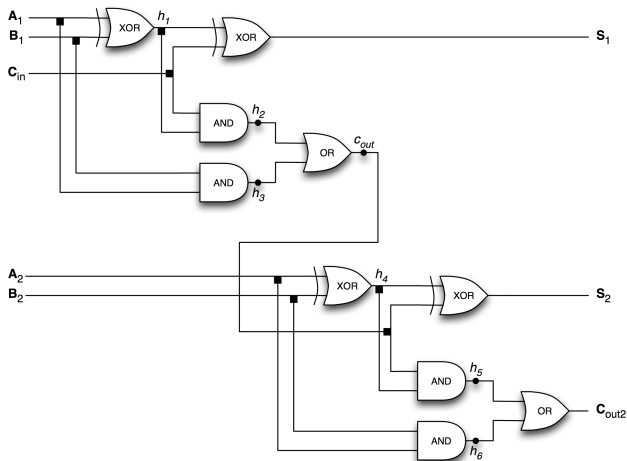
**Question 4:** calculate all possible input / output combinations

*fa*

**Question 5:** Which outputs exist (dependent on  $c_{in}$ )

$\exists a \exists b (fa)$

- Two full adders combined to a 2-bit adder



$$\begin{aligned}
 twofas := & \exists h_1 \exists h_2 \exists h_3 \exists c_{out} \exists h_4 \exists h_5 \exists h_6 \\
 (fa \wedge & (h_4 = xor(a_2, b_2) \wedge s_2 = xor(c_{out}, h_4) \wedge h_5 = (h_4 \& c_{out}) \wedge \\
 & h_6 = (a_2 \& b_2) \wedge c_{out2} = (h_5 \vee h_6)))
 \end{aligned}$$

**Question 6:** calculate all possible input / output combinations

*twofas*

**Question 7:** Outputs dependent on  $a$  and  $a_2$

$\exists b \exists b_2 \exists c_{in}$  (*twofas*)

**Question 8:** Calculate with  $a = a_2 = b = b_2 = 1$  and  $c_{in} = 0$

$\exists a \exists a_2 \exists c_{in} \exists b \exists b_2$  (*twofas*)

**Question 9:** Calculate with  $a = a_2 = b = b_2 = 1$  dependent on  $c_{in} = 0$

$\exists a \exists a_2 \exists b \exists b_2$  (*twofas*)

Possible enhancements:

- Special branching heuristics for free variables
- Mark free variables which affected the result
- More powerful reductions after assigning the free variables

And of course...

- Parallelization