A decision procedure for quantified boolean formulae

Author(s):
Plaisted, David; Biere, Armin

Publication Date:
2000

Permanent Link:

https://doi.org/10.3929/ethz-a-004242435 →

A Decision Procedure for Quantified Boolean Formulae

`http://www.cs.unc.edu/~plaisted/pat/qsat.html`

David Plaisted

University of North Carolina, USA

Armin Biere (*Speaker*)

ETH Zürich, Switzerland

SAT 2000, Renesse, The Netherlands

BDDs vs SAT

Comparison of two implementations of Adders

#Bits	SMV	SATO	GRASP
2	0.1 sec	0.0 sec	0.0 sec
4	0.1 sec	0.0 sec	0.0 sec
8	0.1 sec	0.1 sec	0.3 sec
16	0.1 sec	0.9 sec	2.5 sec
32	0.2 sec	201.8 sec	29.9 sec
64	0.5 sec	—	320.2 sec
128	1.3 sec	—	—
256	5.2 sec	—	—

Often BDDs are much faster than SAT procedures!

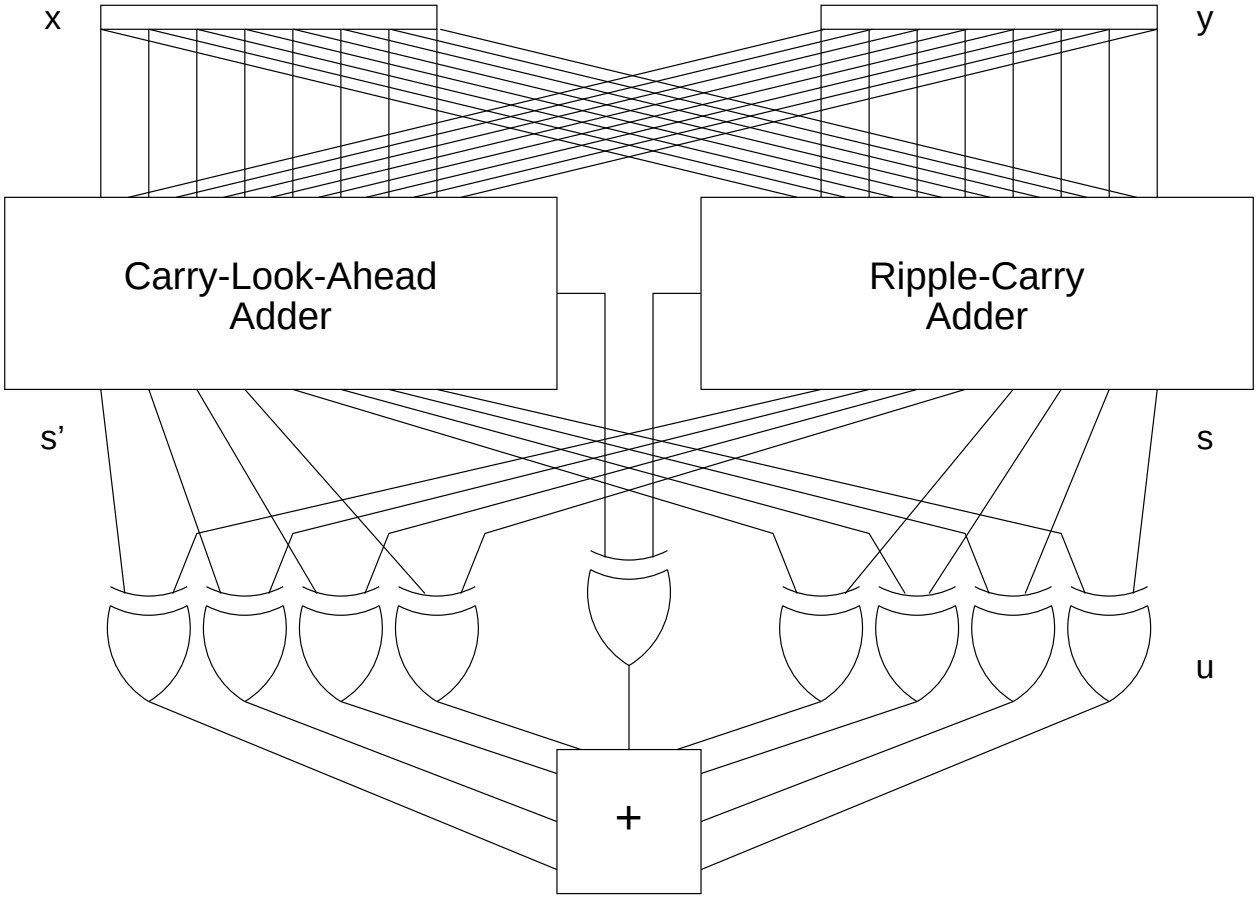
Specification of an 8 Bit Adder

Inputs:	$x = (x_7, \ldots, x_0)$	first operand	$\in \{0, \ldots, 255\}$
	$y = (y_7, \ldots, y_0)$	second operand	$\in \{0, \ldots, 255\}$
Outputs:	$s = (s_7, \ldots, s_0)$	sum of x and y	$\in \{0, \ldots, 255\}$
	c	carry-out bit	$\in \{0, 1\}$

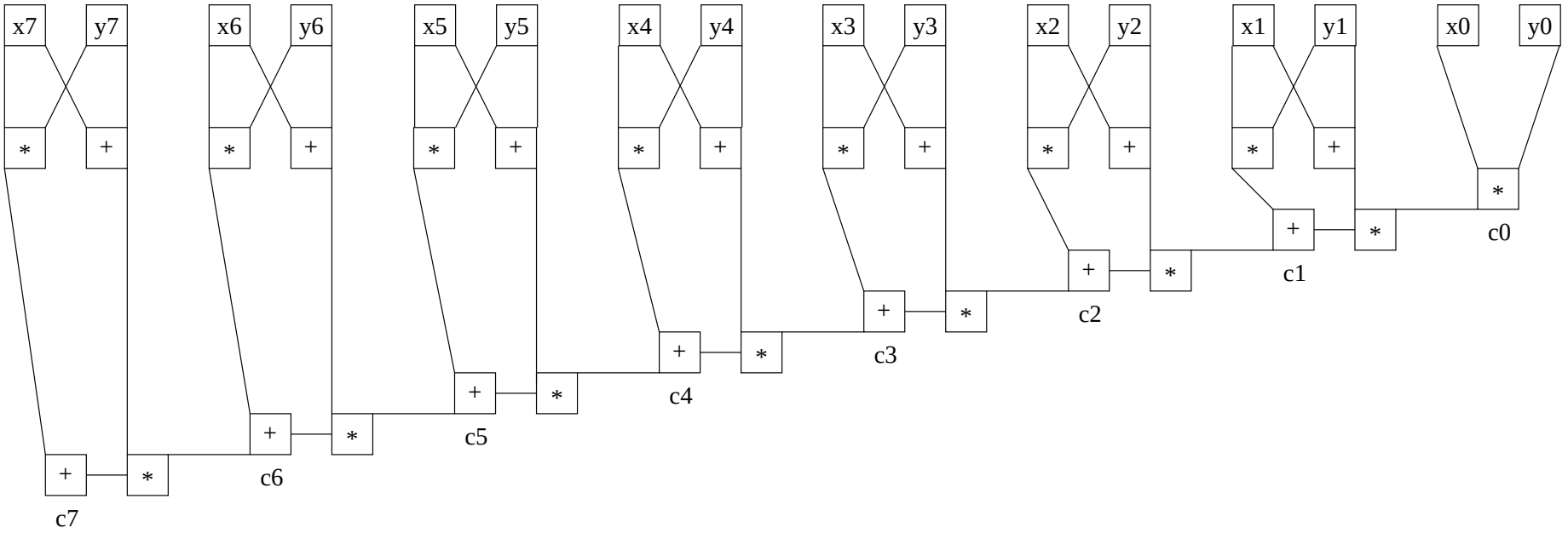
Spec:

$$s = (x + y) \bmod 256$$
$$c = (x + y)/256$$

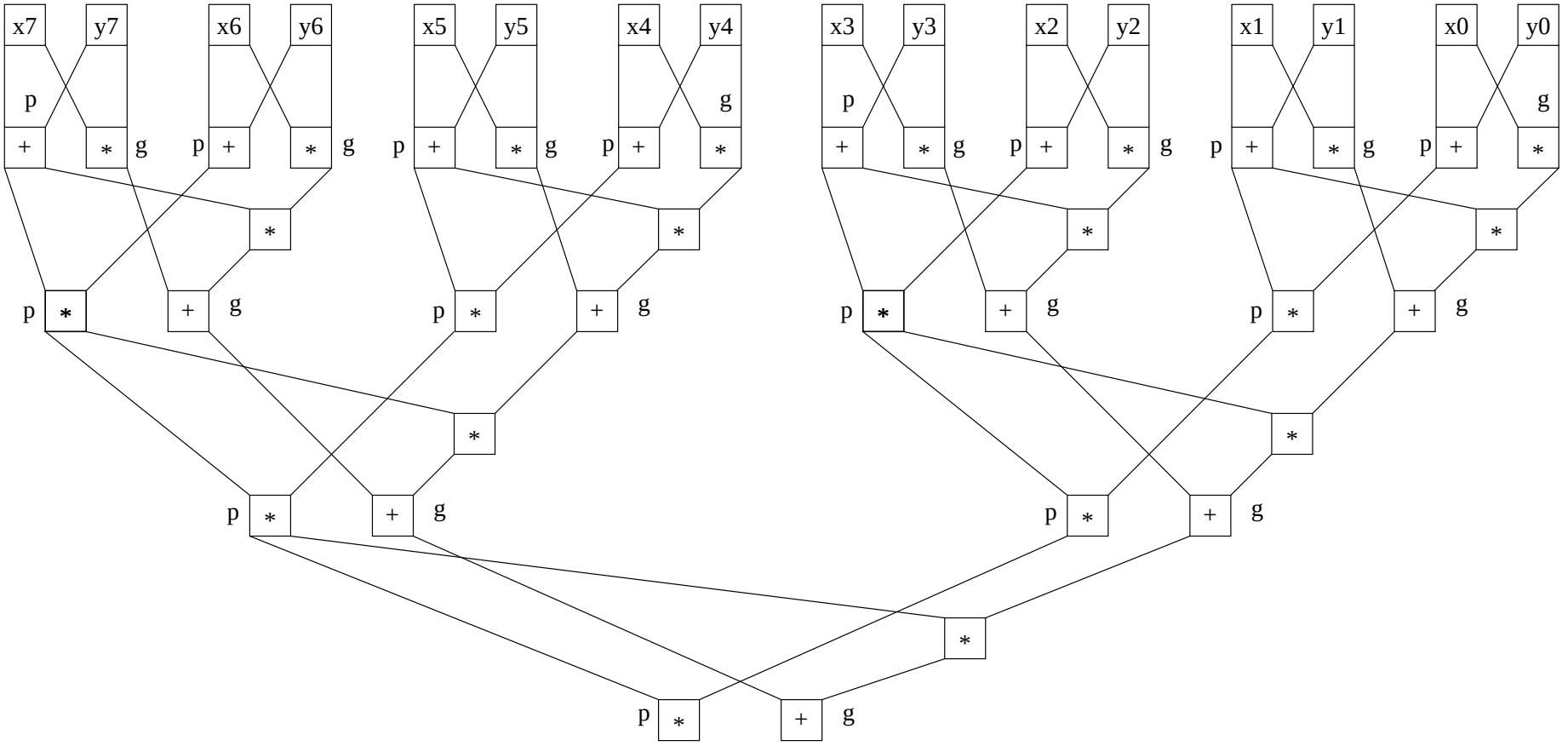
Comparison of Adders



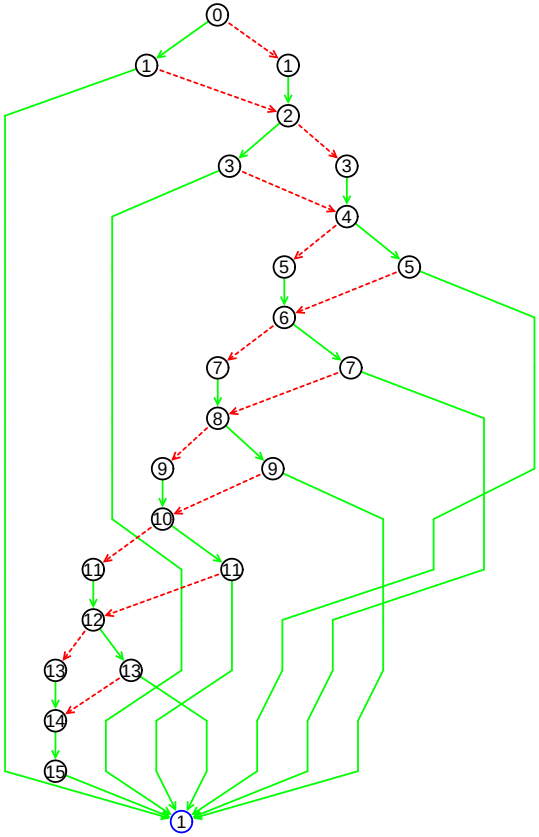
Carry-Out of Ripple-Carry Adder



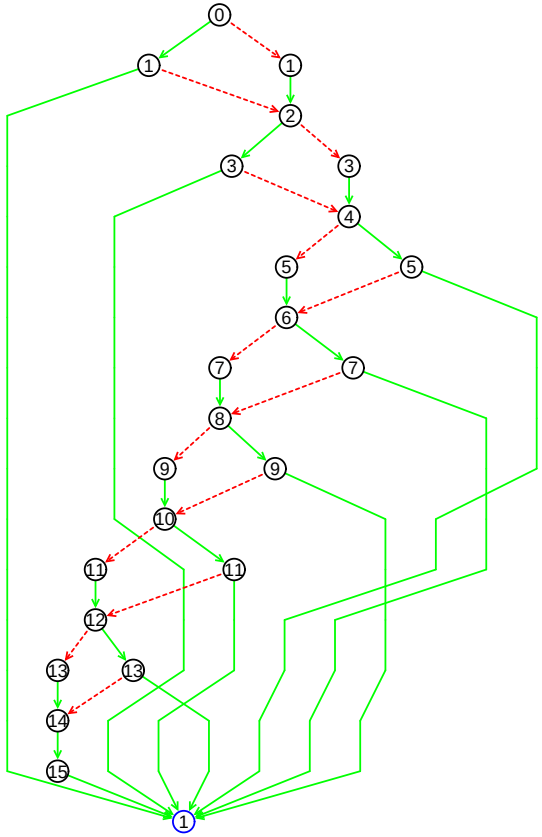
Carry-Out of Carry-Look-Ahead Adder



BDDs for Comparing Carries

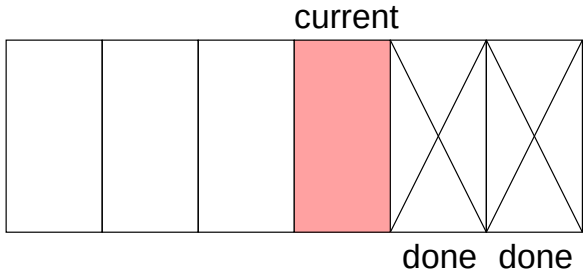


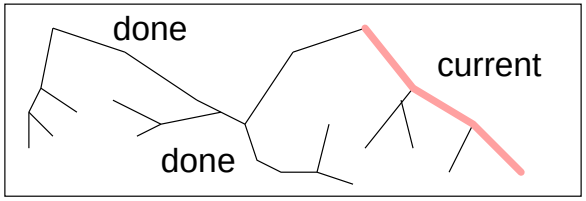
BDD for Carry-Out of
Carry-Look-Ahead Adder



BDD for Carry-Out of
Ripple-Carry Adder

BDDs vs SAT





BDDs

- start at the inputs
- build BDDs from the inside
- maximal caching

SAT

- no explicit search direction
- polynomial space
- clause recording

Simplification Relation $\Rightarrow^*$

$$\begin{aligned} \boxed{F(\exists Z\ [A])} &\equiv F(\exists X\ [\exists Y\ [A]]) \\ &\equiv F(\exists X\ [\exists Y\ [A_1 \wedge A_2]]) \\ &\equiv F(\exists X\ [A_1 \wedge \exists Y\ [A_2]]) \equiv \boxed{F(\exists X\ [A_1 \wedge A_2'])} \end{aligned}$$

where $A_2' \equiv \exists Y\ [A_2]$ lacks the $\exists Y$ quantifiers

$$\boxed{F(\exists Z\ [A])} \Rightarrow \boxed{F(\exists X\ [A_1 \wedge A_2'])}$$

Let F be closed QBF formula: $F \Rightarrow^* \mathit{true}$ or $F \Rightarrow^* \mathit{false}$.

qsat, Algorithm

procedure **qsat**(F);

find F' such that $F \Rightarrow^* F'$ and F' has no quantifiers in it;

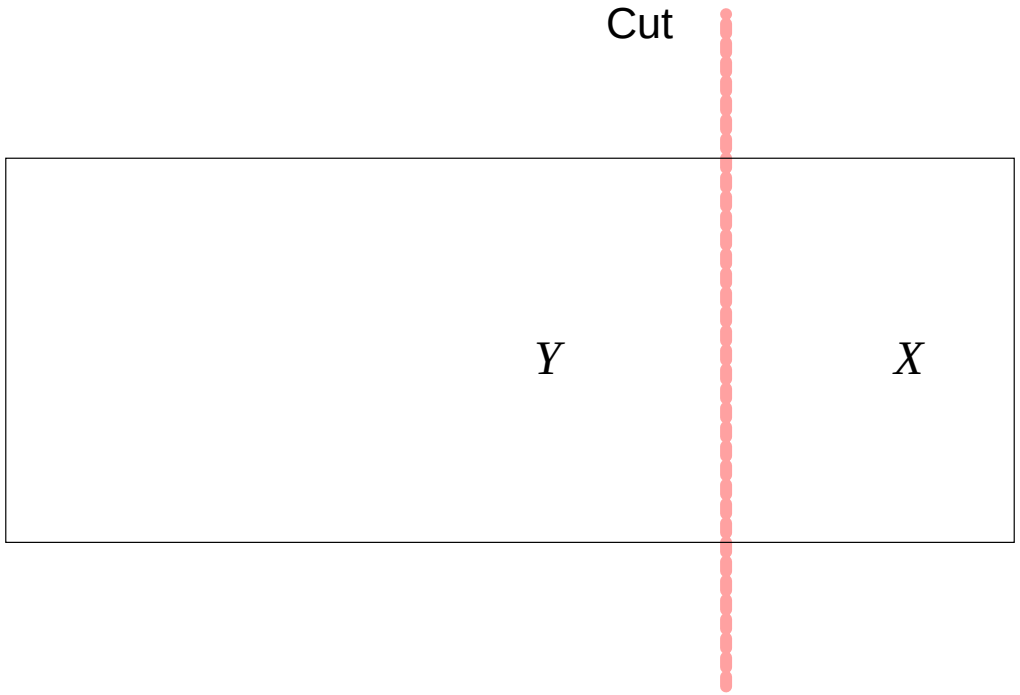
if F' is *true then return* $\boxed{satisfiable}$

elif F' is *false then return* $\boxed{unsatisfiable}$

else return $\boxed{error}$ *fi*

end **qsat**;

Cuts



long and thin formulae allow *small cuts*!

Local and Non-Local Variables

Let A be a subformula of F.

Local Variables of A

all variables of A that occur *exclusively* in A

Non-Local Variables of A

all variables of F that are not local to A

QSAT, Intuition

Basic Step:

1. find subformula A that contains a set of *local variables* Y
2. enumerate assignments $I, J \ldots$ of *non-local variables* X
3. if $A|_I$ is unsatisfiable generate clause to represent $\neg I$
4. replace A by the conjunction A' of all generated clauses

Iteratively quantify out local variables!

simp, Algorithm

procedure **simp**(A, I);

if **unsat**$(A|_I) = \mathit{true}$ *then return* $\boxed{(\neg d_1 \vee \neg d2 \vee \ldots \vee \neg d_n)}$

where $D = \{d_1, d_2, \ldots, d_n\}$ is a subset of I such that $A|_D$ is *unsatisfiable*

elif **taut**$(A|_I) = \mathit{true}$ *then return* $\boxed{\mathit{true}}$

else

let y be some free variable in A such that neither y nor $\neg y$ occurs in I;

let A_1 be **simp**$(A, I \cup \{y\})$;

let A_2 be **simp**$(A, I \cup \{\neg y\})$;

return $\boxed{A_1 \wedge A_2}$

fi

end **simp**;

QBF Example

$$\forall x,y\,[\quad \ldots \quad \exists z\,[(x \vee y \vee z) \wedge (x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)] \quad \ldots \quad]$$

Eliminate innermost quantifiers: $Y = \{z\}, \qquad X = \{x,y\}.$

Assume	$x,\ \ y$	$\Rightarrow$	$(\neg x \vee \neg y)$
Assume	$x,\ \neg y$	$\Rightarrow$	*true*
Assume	$\neg x,\ \ y$	$\Rightarrow$	*true*
Assume	$\neg x,\ \neg y$	$\Rightarrow$	$(x \vee y)$

$$\forall x,y\,[\quad \ldots \qquad (\neg x \vee \neg y) \wedge (x \vee y) \qquad \ldots \quad]$$

Propositional Example (I)

Ripple Adder $(s_0 = (x_0 \neq y_0)) \wedge (c_0 = (x_0 \wedge y_0)) \wedge (s_1 = (x_1 = y_1 = c_0)) \wedge \ldots$

XOR array $\ldots \wedge (u_0 = (s_0 \neq s'_0)) \wedge \ldots \wedge (u_7 = (s_7 \neq s'_7)) \wedge (u_8 = (c_7 \neq g_7)) \wedge \ldots$

Disjunction $\ldots \wedge (u_0 \vee u_1 \vee \ldots \vee u_7 \vee u_8) \wedge \ldots$

Look Ahead Adder $\ldots \wedge (s'_0 = (x_0 \neq y_0)) \wedge (g_0 = (x_0 \wedge y_0)) \wedge (s'_1 = (x_1 = y_1 = g_0))$

$$Y = \{x_0, y_0, s_0, s'_0\}, \quad X = \{c_0, g_0, u_0\}$$

Propositional Example (II)

$$Y = \{x_0, y_0, s_0, s_0'\}, \quad X = \{c_0, g_0, u_0\}$$

$$\exists\, x_0, y_0, s_0, s_0' \,[\quad (s_0 = (x_0 \neq y_0)) \wedge (c_0 = (x_0 \wedge y_0)) \quad \wedge$$
$$(u_0 = (s_0 \neq s_0')) \quad \wedge$$
$$(s_0' = (x_0 \neq y_0)) \wedge (g_0 = (x_0 \wedge y_0)) \quad]$$

Assume	$u_0,$	$\ldots$	$\Rightarrow$	$\neg u_0$
Assume	$\neg u_0,$	$\neg c_0, \neg g_0$	$\Rightarrow$	$true$
Assume	$\neg u_0,$	$\neg c_0, \quad g_0$	$\Rightarrow$	$(c_0 \vee \neg g_0)$
Assume	$\neg u_0,$	$c_0, \neg g_0$	$\Rightarrow$	$(\neg c_0 \vee g_0)$
Assume	$\neg u_0,$	$c_0, \quad g_0$	$\Rightarrow$	$true$

$$\neg u_0 \wedge (c_0 \vee \neg g_0) \wedge (\neg c_0 \vee g_0)$$

Related Work

- [Kühlmann et.al.] Equivalence Checking
- ATPG: directed search for SAT problems in circuit design
- [Rintanen], [Cadoli et.al.], [Stålmark] QBF decision procedures
- [Biere et.al.] Bounded Model Checking
- [Williams et.al.] BEDs + SAT + BDDs

Conclusion

- decision procedure for QBF
- bottom-up instead of top-down
- combines features of BDDs with SAT
- works best for long and thin formulae
- no (working) implementation yet