CEGAR-based EF Synthesis of Boolean Functions with an Application to Circuit Rectification

Heinz Riener, Rüdiger Ehlers, and Goerschwin Fey
German Aerospace Center, Bremen, Germany
DFKI GmbH, Germany
University of Bremen, Germany

DLR

CEGAR-based Exist-Forall Synthesis of Boolean functions

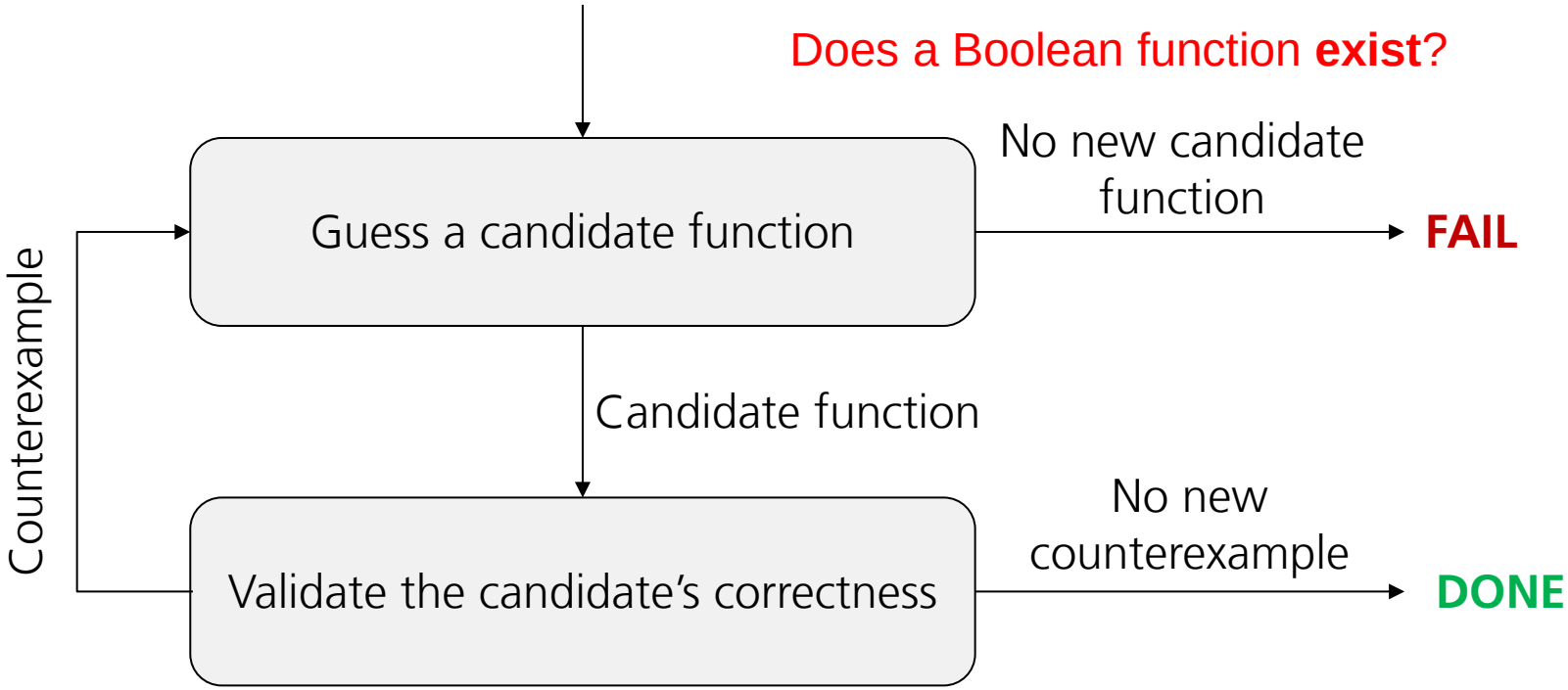
- Does a Boolean function F exist with respect to a correctness specification Q such that $Q\big(X, F(X)\big)$ holds for all possible values of X?

$$\exists F: \forall X: Q\big(X, F(X)\big)$$

- **Challenge:**
  - Not a SAT-problem: Exist-forall (EF) quantifier-alternation
  - Not a QBF-problem: F is a second-order variable (domain is the set of all Boolean functions)

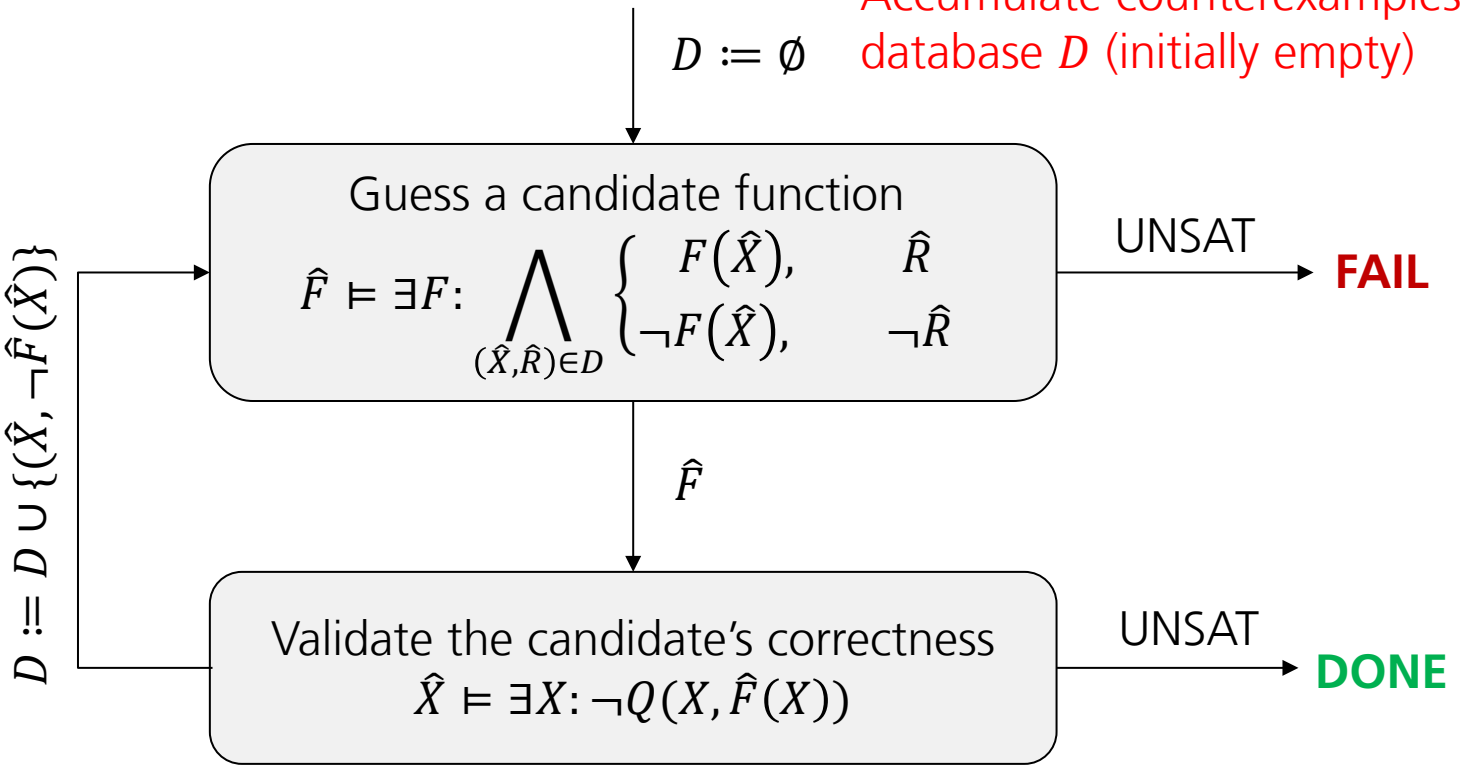
- **Contribution:**
  - CEGAR-based loop for computing a model for F in a normal form representation of bounded size using incremental Boolean learning
  - Demonstration of the approach in the context of circuit rectification, e.g., applicable for ECO-synthesis

CEGAR-based EF Synthesis of Boolean functions

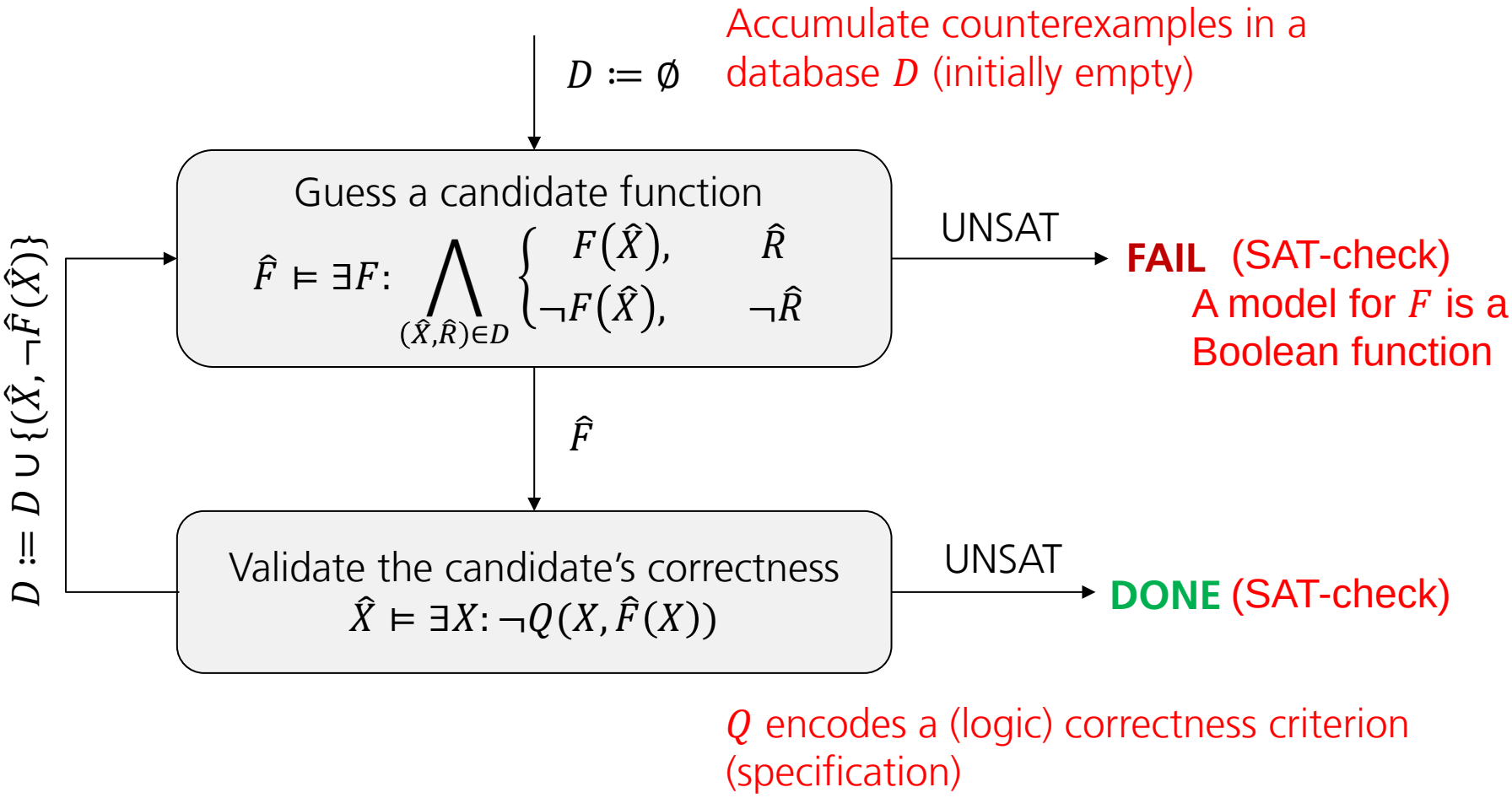


Is it valid with respect to a specification **for all** possible inputs?

CEGAR-based EF Synthesis of Boolean functions



CEGAR-based EF Synthesis of Boolean functions



Computing “function models” using Boolean SAT

- Many techniques are possible – we use **bounded synthesis + Boolean learning**
- Synthesizes a Boolean function in a normal form representation from counterexamples (= input-output samples)

1. Start from a normal form representation of bounded size with open parameters to be determined
2. Use the counterexamples to infer the parameters
3. Refine when new counterexamples are provided

- **SAT**: parameters have been found to “concretize” the normal form expression
- **UNSAT**: no parameters exist (within the bound)
  - Issue: either no function at all exists or the bounds need to be relaxed
- **Incremental approach**: keeps the SAT-solver’s state alive, add constraints for new counterexamples

Computing “function models” using Boolean SAT

- Consider **Sum-Of-Products (SOP**) / **Disjunctive Normal Form (DNF)**
  - Other normal forms are possible too, but different formalization required

- Suppose that $F(X)$ is a Boolean function in SOP to be determined over variables $X := x_1, \ldots, x_n$ and restricted to at most m cubes

- Construct a SAT-problem over $2nm$ Boolean variables: $p_{j,l}$ and $q_{j,l}$ for $1 \leq j \leq m$ and $1 \leq l \leq n$, where

$$p_{j,l} := \begin{cases} 1, & x_l \text{ appears in cube } j \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad q_{j,l} := \begin{cases} 1, & \neg x_l \text{ appears in cube } j \\ 0, & \text{otherwise.} \end{cases}$$

- If $p_{j,l} = 0$ and $q_{j,l} = 0$ then variable x_l does not appear in cube j
- If $p_{j,l} = 1$ and $q_{j,l} = 1$ then cube j cancels out

Computing "function models" using Boolean SAT

- Suppose that $D \coloneqq (\hat{X}_1, \hat{R}_1), \ldots, (\hat{X}_p, \hat{R}_p)$ is a database of input-output samples (= a partially-specified truth table), assignments $P \coloneqq p_{1,1}, \ldots, p_{m,n}$ and $Q \coloneqq q_{1,1}, \ldots, q_{m,n}$ are obtained by solving

$$\hat{P}, \hat{Q} \vDash \bigwedge_{(\hat{X},\hat{R}) \in D} \begin{cases} N(\hat{X}, P, Q), & \hat{R} = 0 \\ P(\hat{X}, P, Q), & \hat{R} = 1 \end{cases}$$

with

$$N(x_1, \ldots, x_n; p_{1,1}, \ldots, p_{m,n}; q_{1,1}, \ldots, q_{m,n}) \coloneqq \bigwedge_{j=1}^{m} \bigvee_{l=1}^{n} \mathrm{ITE}(x_l, q_{j,l}, p_{j,l})$$

Each cube must disagree with at least one value of the sample

and

$$P(x_1, \ldots, x_n; p_{1,1}, \ldots, p_{m,n}; q_{1,1}, \ldots, q_{m,n}) \coloneqq \exists z_1, \ldots, z_m : \left(\bigvee_{j=1}^{m} z_j\right) \wedge \left(\bigwedge_{j=1}^{m} \bigwedge_{l=1}^{n} \neg z_j \vee \neg \mathrm{ITE}(x_l, q_{j,l}, p_{j,l})\right)$$

At least one cube must be satisfied

For all satisfied cubes, p and q have to be consistent with values of the sample

Computing “function models” using Boolean SAT

- Concrete assignments to $P \coloneqq p_{1,1}, \dots, p_{m,n}$ and $Q \coloneqq q_{1,1}, \dots, q_{m,n}$ concretize the generic expression

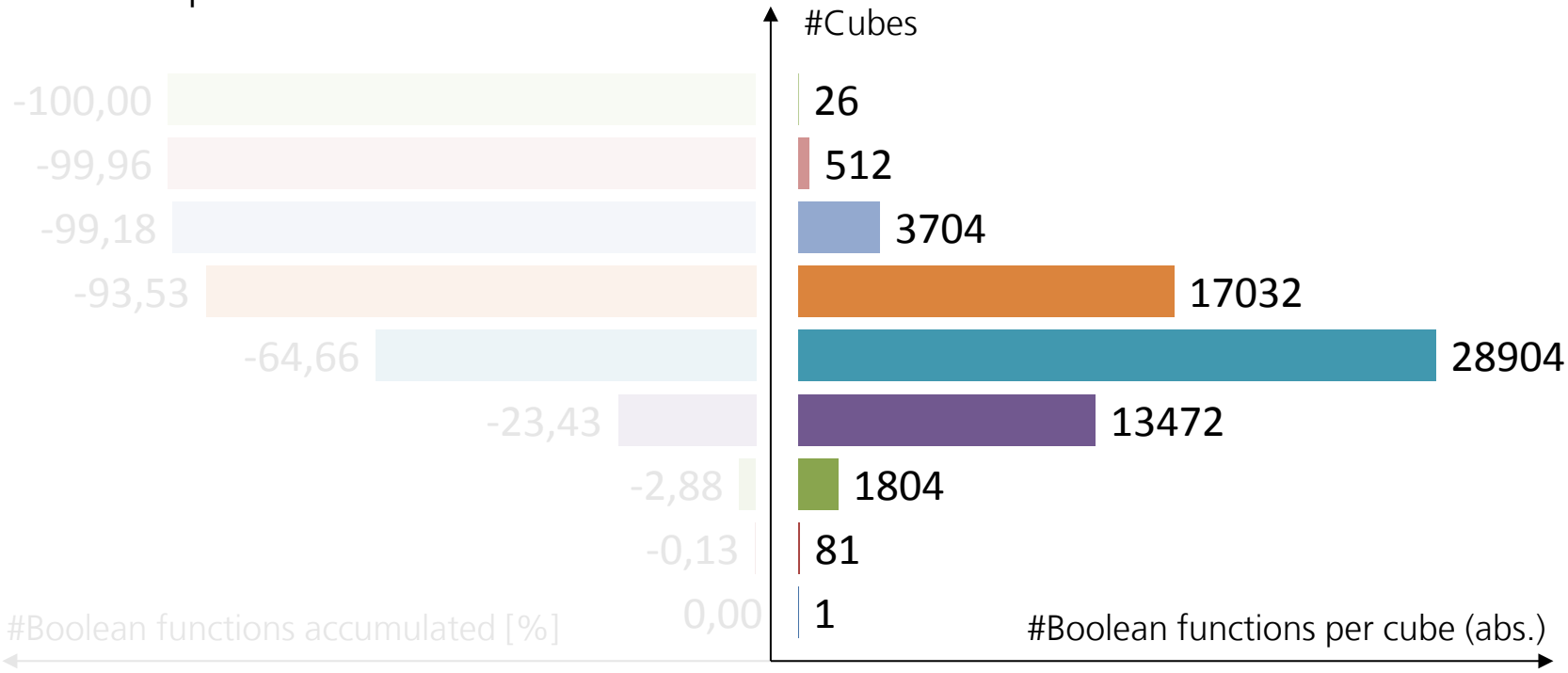
$$F\left(x_1, \dots, x_n; p_{1,1}, \dots, p_{m,n}; q_{1,1}, \dots, q_{m,n}\right) \coloneqq \bigvee_{j=1}^{m} \bigwedge_{l=1}^{n} \mathrm{ITE}\left(p_{j,l}, x_l, \mathrm{true}\right) \wedge \mathrm{ITE}\left(q_{j,l}, \neg x_l, \mathrm{true}\right)$$

  to an SOP-expression

ITE-expressions are simplified by constant propagation in a preprocessing step

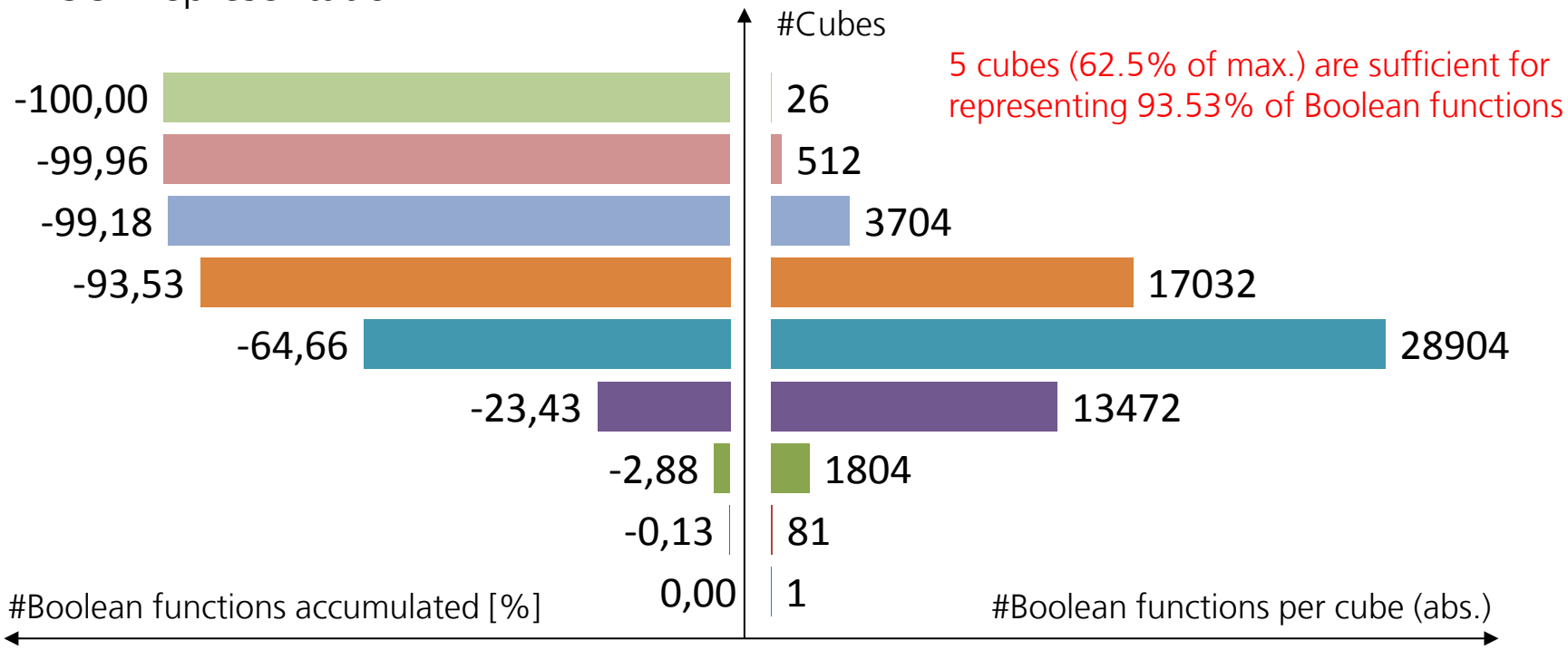
SOP representation of Boolean functions

- Consider completely-specified Boolean functions over 4 variables ($2^{2^4} = 65536$) in SOP representation



SOP representation of Boolean functions

- Consider completely-specified Boolean functions over 4 variables ($2^{2^4} = 65536$) in SOP representation



Incremental learning of Boolean functions by example

Hidden Boolean function

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$	
	0	0	0	0	**0**	0x4
	0	0	0	1	**0**	
	0	0	1	0	**1**	
	0	0	1	1	**0**	
	0	1	0	0	**1**	0x5
	0	1	0	1	**0**	
	0	1	1	0	**1**	
	0	1	1	1	**0**	
	1	0	0	0	**1**	0x5
	1	0	0	1	**0**	
	1	0	1	0	**1**	
	1	0	1	1	**0**	
	1	1	0	0	**1**	0x1
	1	1	0	1	**0**	
	1	1	1	0	**0**	
	1	1	1	1	**0**	

Evolution of SOP-expression:

id	SOP

- The Boolean function 0x1554 shall be learned

Incremental learning of Boolean functions by example

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$
	0	0	0	0	**0**
	0	0	0	1	**0**
	0	0	1	0	**1**
	0	0	1	1	**0**
	0	1	0	0	**1**
1.	**0**	**1**	**0**	**1**	**0**
	0	1	1	0	**1**
	0	1	1	1	**0**
	1	0	0	0	**1**
	1	0	0	1	**0**
	1	0	1	0	**1**
	1	0	1	1	**0**
	1	1	0	0	**1**
	1	1	0	1	**0**
	1	1	1	0	**0**
	1	1	1	1	**0**

Evolution of SOP-expression:

id	SOP

● Fixed sample

Incremental learning of Boolean functions by example

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$
	0	0	0	0	0
	0	0	0	1	0
	0	0	1	0	0
	0	0	1	1	0
	0	1	0	0	0
1.	**0**	**1**	**0**	**1**	**0**
	0	1	1	0	0
	0	1	1	1	0
	1	0	0	0	0
	1	0	0	1	0
	1	0	1	0	0
	1	0	1	1	0
	1	1	0	0	0
	1	1	0	1	0
	1	1	1	0	0
	1	1	1	1	0

Evolution of SOP-expression:

id	SOP
1.	$false$

- Correctly chosen
- Erroneously chosen
- Fixed sample

Incremental learning of Boolean functions by example

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$
	0	0	0	0	0
	0	0	0	1	0
2.	**0**	**0**	**1**	**0**	**1**
	0	0	1	1	0
	0	1	0	0	0
1.	**0**	**1**	**0**	**1**	**0**
	0	1	1	0	0
	0	1	1	1	0
	1	0	0	0	0
	1	0	0	1	0
	1	0	1	0	0
	1	0	1	1	0
	1	1	0	0	0
	1	1	0	1	0
	1	1	1	0	0
	1	1	1	1	0

Evolution of SOP-expression:

id	SOP
1.	$false$

- Correctly chosen
- Erroneously chosen
- Fixed sample

Incremental learning of Boolean functions by example

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$
	0	0	0	0	1
	0	0	0	1	0
2.	**0**	**0**	**1**	**0**	**1**
	0	0	1	1	0
	0	1	0	0	1
1.	**0**	**1**	**0**	**1**	**0**
	0	1	1	0	1
	0	1	1	1	0
	1	0	0	0	1
	1	0	0	1	0
	1	0	1	0	1
	1	0	1	1	0
	1	1	0	0	1
	1	1	0	1	0
	1	1	1	0	1
	1	1	1	1	0

Evolution of SOP-expression:

id	SOP
1.	$false$
2.	$(\neg x_1)$

- Correctly chosen
- Erroneously chosen
- Fixed sample

Incremental learning of Boolean functions by example

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$
3.	**0**	**0**	**0**	**0**	**0**
	0	0	0	1	**0**
2.	**0**	**0**	**1**	**0**	**1**
	0	0	1	1	**0**
	0	1	0	0	**1**
1.	**0**	**1**	**0**	**1**	**0**
	0	1	1	0	**1**
	0	1	1	1	**0**
	1	0	0	0	**1**
	1	0	0	1	**0**
	1	0	1	0	**1**
	1	0	1	1	**0**
	1	1	0	0	**1**
	1	1	0	1	**0**
	1	1	1	0	**1**
	1	1	1	1	**0**

Evolution of SOP-expression:

id	SOP
1.	$false$
2.	$(\neg x_1)$

- Correctly chosen
- Erroneously chosen
- Fixed sample

Incremental learning of Boolean functions by example

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$
3.	**0**	**0**	**0**	**0**	**0**
	0	0	0	1	**0**
2.	**0**	**0**	**1**	**0**	**1**
	0	0	1	1	**1**
	0	1	0	0	**1**
1.	**0**	**1**	**0**	**1**	**0**
	0	1	1	0	**1**
	0	1	1	1	**1**
	1	0	0	0	**0**
	1	0	0	1	**0**
	1	0	1	0	**1**
	1	0	1	1	**1**
	1	1	0	0	**0**
	1	1	0	1	**0**
	1	1	1	0	**1**
	1	1	1	1	**1**

Evolution of SOP-expression:

id	SOP
1.	$false$
2.	$(\neg x_1)$
3.	(x_2)

- Correctly chosen
- Erroneously chosen
- Fixed sample

Incremental learning of Boolean functions by example

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$
3.	**0**	**0**	**0**	**0**	**0**
	0	0	0	1	**0**
2.	**0**	**0**	**1**	**0**	**1**
4.	**0**	**0**	**1**	**1**	**0**
	0	1	0	0	**1**
1.	**0**	**1**	**0**	**1**	**0**
	0	1	1	0	**1**
	0	1	1	1	**1**
	1	0	0	0	**0**
	1	0	0	1	**0**
	1	0	1	0	**1**
	1	0	1	1	**1**
	1	1	0	0	**0**
	1	1	0	1	**0**
	1	1	1	0	**1**
	1	1	1	1	**1**

Evolution of SOP-expression:

id	SOP
1.	$false$
2.	$(\neg x_1)$
3.	(x_2)

- Correctly chosen
- Erroneously chosen
- Fixed sample

Incremental learning of Boolean functions by example

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$
3.	**0**	**0**	**0**	**0**	**0**
	0	0	0	1	**0**
2.	**0**	**0**	**1**	**0**	**1**
4.	**0**	**0**	**1**	**1**	**0**
	0	1	0	0	**0**
1.	**0**	**1**	**0**	**1**	**0**
	0	1	1	0	**1**
	0	1	1	1	**0**
	1	0	0	0	**0**
	1	0	0	1	**0**
	1	0	1	0	**1**
	1	0	1	1	**0**
	1	1	0	0	**0**
	1	1	0	1	**0**
	1	1	1	0	**1**
	1	1	1	1	**0**

Evolution of SOP-expression:

id	SOP
1.	$false$
2.	$(\neg x_1)$
3.	(x_2)
4.	$(\neg x_1 \wedge x_2)$

● Correctly chosen
● Erroneously chosen
● Fixed sample

Incremental learning of Boolean functions by example

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$
3.	**0**	**0**	**0**	**0**	**0**
	0	0	0	1	**0**
2.	**0**	**0**	**1**	**0**	**1**
4.	**0**	**0**	**1**	**1**	**0**
5.	**0**	**1**	**0**	**0**	**1**
1.	**0**	**1**	**0**	**1**	**0**
	0	1	1	0	**1**
	0	1	1	1	**0**
	1	0	0	0	**0**
	1	0	0	1	**0**
	1	0	1	0	**1**
	1	0	1	1	**0**
	1	1	0	0	**0**
	1	1	0	1	**0**
	1	1	1	0	**1**
	1	1	1	1	**0**

Evolution of SOP-expression:

id	SOP
1.	$false$
2.	$(\neg x_1)$
3.	(x_2)
4.	$(\neg x_1 \wedge x_2)$

● Correctly chosen
● Erroneously chosen
● Fixed sample

Incremental learning of Boolean functions by example

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$
3.	**0**	**0**	**0**	**0**	**0**
	0	0	0	1	**0**
2.	**0**	**0**	**1**	**0**	**1**
4.	**0**	**0**	**1**	**1**	**0**
5.	**0**	**1**	**0**	**0**	**1**
1.	**0**	**1**	**0**	**1**	**0**
	0	1	1	0	**1**
	0	1	1	1	**0**
	1	0	0	0	**0**
	1	0	0	1	**0**
	1	0	1	0	**1**
	1	0	1	1	**0**
	1	1	0	0	**0**
	1	1	0	1	**0**
	1	1	1	0	**1**
	1	1	1	1	**0**

Evolution of SOP-expression:

id	SOP
1.	$false$
2.	$(\neg x_1)$
3.	(x_2)
4.	$(\neg x_1 \wedge x_2)$
5.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$

● Correctly chosen
● Erroneously chosen
● Fixed sample

Incremental learning of Boolean functions by example

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$
3.	**0**	**0**	**0**	**0**	**0**
	0	0	0	1	**0**
2.	**0**	**0**	**1**	**0**	**1**
4.	**0**	**0**	**1**	**1**	**0**
5.	**0**	**1**	**0**	**0**	**1**
1.	**0**	**1**	**0**	**1**	**0**
	0	1	1	0	**1**
	0	1	1	1	**0**
6.	**1**	**0**	**0**	**0**	**1**
	1	0	0	1	**0**
	1	0	1	0	**1**
	1	0	1	1	**0**
	1	1	0	0	**0**
	1	1	0	1	**0**
	1	1	1	0	**1**
	1	1	1	1	**0**

Evolution of SOP-expression:

id	SOP
1.	$false$
2.	$(\neg x_1)$
3.	(x_2)
4.	$(\neg x_1 \wedge x_2)$
5.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$

- Correctly chosen
- Erroneously chosen
- Fixed sample

Incremental learning of Boolean functions by example

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$
3.	**0**	**0**	**0**	**0**	**0**
	0	0	0	1	**0**
2.	**0**	**0**	**1**	**0**	**1**
4.	**0**	**0**	**1**	**1**	**0**
5.	**0**	**1**	**0**	**0**	**1**
1.	**0**	**1**	**0**	**1**	**0**
	0	1	1	0	**1**
	0	1	1	1	**0**
6.	**1**	**0**	**0**	**0**	**1**
	1	0	0	1	**1**
	1	0	1	0	**1**
	1	0	1	1	**1**
	1	1	0	0	**1**
	1	1	0	1	**1**
	1	1	1	0	**1**
	1	1	1	1	**1**

Evolution of SOP-expression:

id	SOP
1.	$false$
2.	$(\neg x_1)$
3.	(x_2)
4.	$(\neg x_1 \wedge x_2)$
5.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$
6.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3) \vee (x_4)$

- Correctly chosen
- Erroneously chosen
- Fixed sample

Incremental learning of Boolean functions by example

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$
3.	**0**	**0**	**0**	**0**	**0**
	0	0	0	1	**0**
2.	**0**	**0**	**1**	**0**	**1**
4.	**0**	**0**	**1**	**1**	**0**
5.	**0**	**1**	**0**	**0**	**1**
1.	**0**	**1**	**0**	**1**	**0**
	0	1	1	0	**1**
	0	1	1	1	**0**
6.	**1**	**0**	**0**	**0**	**1**
7.	**1**	**0**	**0**	**1**	**0**
	1	0	1	0	**1**
	1	0	1	1	**1**
	1	1	0	0	**1**
	1	1	0	1	**1**
	1	1	1	0	**1**
	1	1	1	1	**1**

Evolution of SOP-expression:

id	SOP
1.	$false$
2.	$(\neg x_1)$
3.	(x_2)
4.	$(\neg x_1 \wedge x_2)$
5.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$
6.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3) \vee (x_4)$

- Correctly chosen
- Erroneously chosen
- Fixed sample

Incremental learning of Boolean functions by example

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$
3.	**0**	**0**	**0**	**0**	**0**
	0	0	0	1	**0**
2.	**0**	**0**	**1**	**0**	**1**
4.	**0**	**0**	**1**	**1**	**0**
5.	**0**	**1**	**0**	**0**	**1**
1.	**0**	**1**	**0**	**1**	**0**
	0	1	1	0	**1**
	0	1	1	1	**0**
6.	**1**	**0**	**0**	**0**	**1**
7.	**1**	**0**	**0**	**1**	**0**
	1	0	1	0	**1**
	1	0	1	1	**0**
	1	1	0	0	**1**
	1	1	0	1	**0**
	1	1	1	0	**1**
	1	1	1	1	**0**

Evolution of SOP-expression:

id	SOP
1.	$false$
2.	$(\neg x_1)$
3.	(x_2)
4.	$(\neg x_1 \wedge x_2)$
5.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$
6.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3) \vee (x_4)$
7.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3) \vee (\neg x_1 \wedge x_4)$

- Correctly chosen
- Erroneously chosen
- Fixed sample

Incremental learning of Boolean functions by example

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$
3.	**0**	**0**	**0**	**0**	**0**
	0	0	0	1	**0**
2.	**0**	**0**	**1**	**0**	**1**
4.	**0**	**0**	**1**	**1**	**0**
5.	**0**	**1**	**0**	**0**	**1**
1.	**0**	**1**	**0**	**1**	**0**
	0	1	1	0	**1**
	0	1	1	1	**0**
6.	**1**	**0**	**0**	**0**	**1**
7.	**1**	**0**	**0**	**1**	**0**
	1	0	1	0	**1**
	1	0	1	1	**0**
	1	1	0	0	**1**
	1	1	0	1	**0**
8.	**1**	**1**	**1**	**0**	**0**
	1	1	1	1	**0**

Evolution of SOP-expression:

id	SOP
1.	$false$
2.	$(\neg x_1)$
3.	(x_2)
4.	$(\neg x_1 \wedge x_2)$
5.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$
6.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3) \vee (x_4)$
7.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3) \vee (\neg x_1 \wedge x_4)$

- Correctly chosen
- Erroneously chosen
- Fixed sample

Incremental learning of Boolean functions by example

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$
3.	**0**	**0**	**0**	**0**	**0**
	0	0	0	1	**0**
2.	**0**	**0**	**1**	**0**	**1**
4.	**0**	**0**	**1**	**1**	**0**
5.	**0**	**1**	**0**	**0**	**1**
1.	**0**	**1**	**0**	**1**	**0**
	0	1	1	0	**1**
	0	1	1	1	**0**
6.	**1**	**0**	**0**	**0**	**1**
7.	**1**	**0**	**0**	**1**	**0**
	1	0	1	0	**1**
	1	0	1	1	**0**
	1	1	0	0	**1**
	1	1	0	1	**0**
8.	**1**	**1**	**1**	**0**	**0**
	1	1	1	1	**0**

Evolution of SOP-expression:

id	SOP
1.	$false$
2.	$(\neg x_1)$
3.	(x_2)
4.	$(\neg x_1 \wedge x_2)$
5.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$
6.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3) \vee (x_4)$
7.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3) \vee (\neg x_1 \wedge x_4)$
8.	$(\neg x_1 \wedge \neg x_2 \wedge x_4) \vee (\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3)$

- Correctly chosen
- Erroneously chosen
- Fixed sample

Incremental learning of Boolean functions by example

Database:

id	$\hat{x}_4$	$\hat{x}_3$	$\hat{x}_2$	$\hat{x}_1$	$\hat{r}$
3.	**0**	**0**	**0**	**0**	**0**
	0	0	0	1	**0**
2.	**0**	**0**	**1**	**0**	**1**
4.	**0**	**0**	**1**	**1**	**0**
5.	**0**	**1**	**0**	**0**	**1**
1.	**0**	**1**	**0**	**1**	**0**
	0	1	1	0	**1**
	0	1	1	1	**0**
6.	**1**	**0**	**0**	**0**	**1**
7.	**1**	**0**	**0**	**1**	**0**
	1	0	1	0	**1**
	1	0	1	1	**0**
	1	1	0	0	**1**
	1	1	0	1	**0**
8.	**1**	**1**	**1**	**0**	**0**
	1	1	1	1	**0**

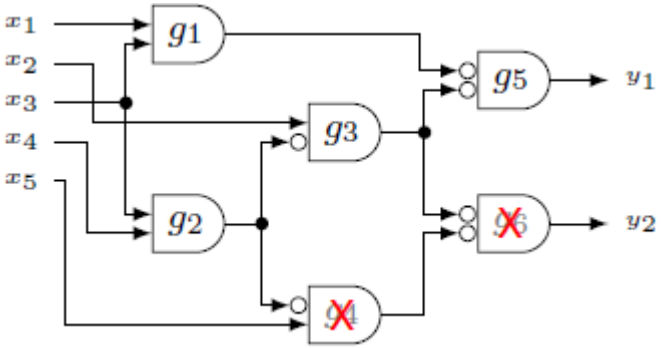
Evolution of SOP-expression:

id	SOP
1.	$false$
2.	$(\neg x_1)$
3.	(x_2)
4.	$(\neg x_1 \wedge x_2)$
5.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$
6.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3) \vee (x_4)$
7.	$(\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3) \vee (\neg x_1 \wedge x_4)$
8.	$(\neg x_1 \wedge \neg x_2 \wedge x_4) \vee (\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3)$

CORRECT! (No new counterexamples)

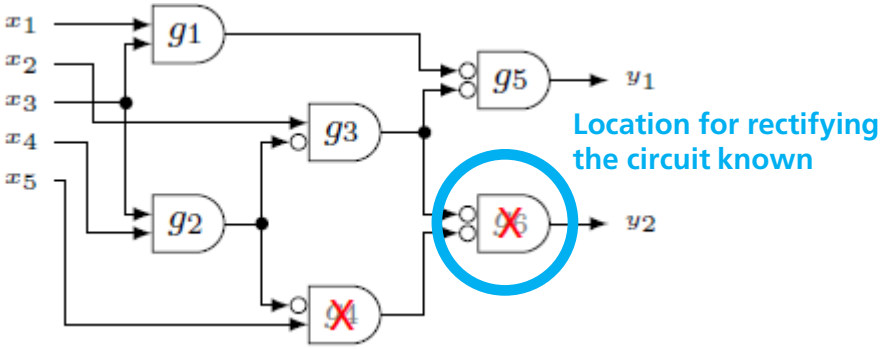
- Correctly chosen
- Erroneously chosen
- Fixed sample

Incremental patch synthesis by example



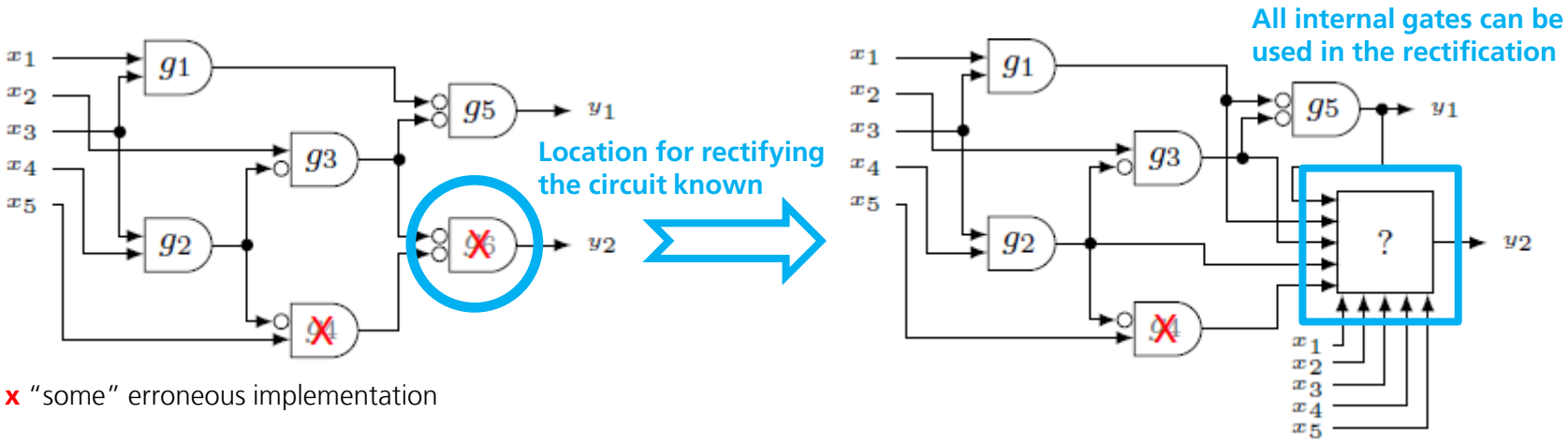
x “some” erroneous implementation

Incremental patch synthesis by example



x “some” erroneous implementation

Incremental patch synthesis by example

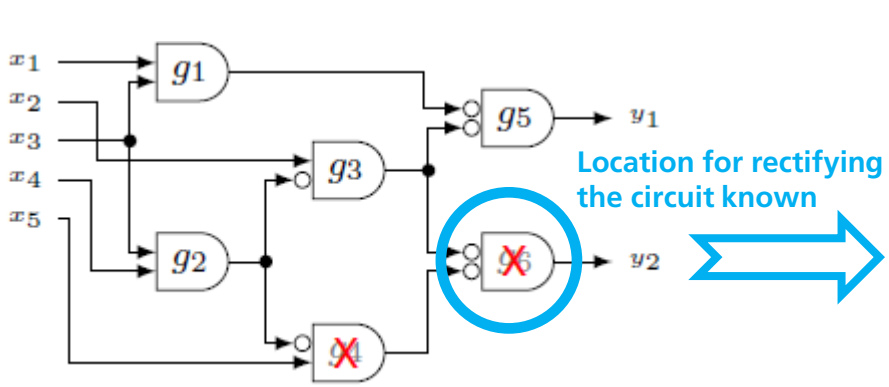


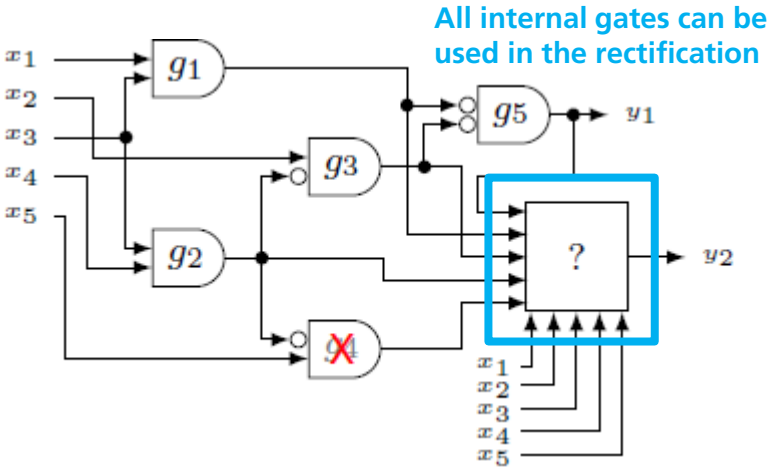
x "some" erroneous implementation

id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6

Patch: ?

Incremental patch synthesis by example





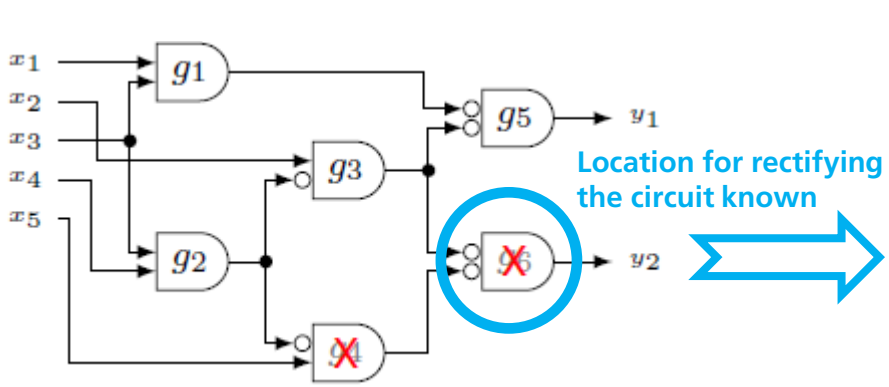
x "some" erroneous implementation

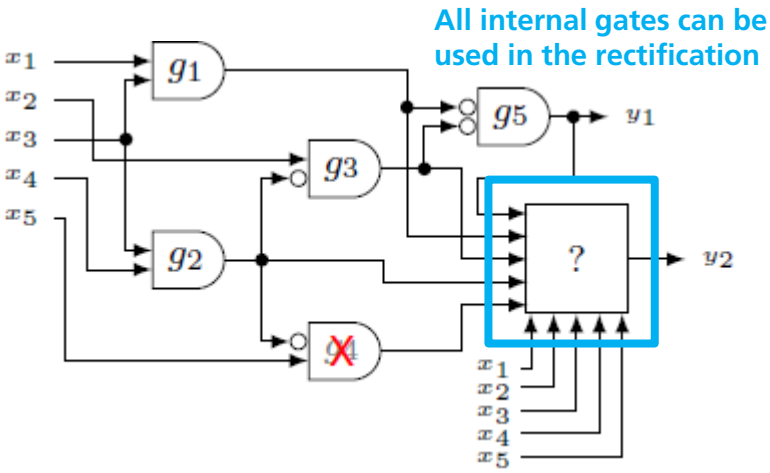
id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0							

1. Counterexample computed with equivalence checking, e.g., ABC

Patch: ?

Incremental patch synthesis by example





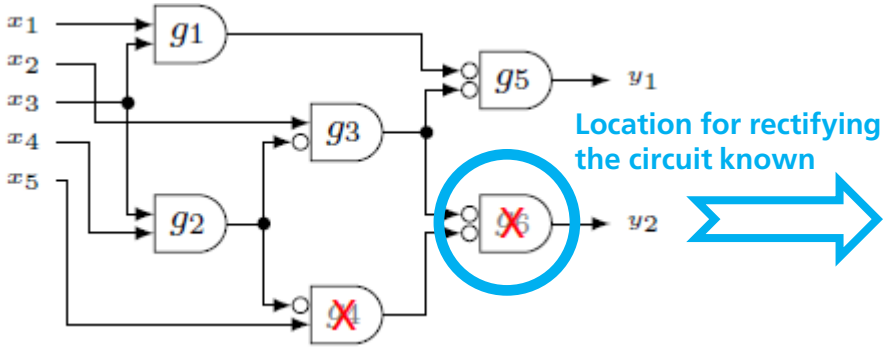
x "some" erroneous implementation

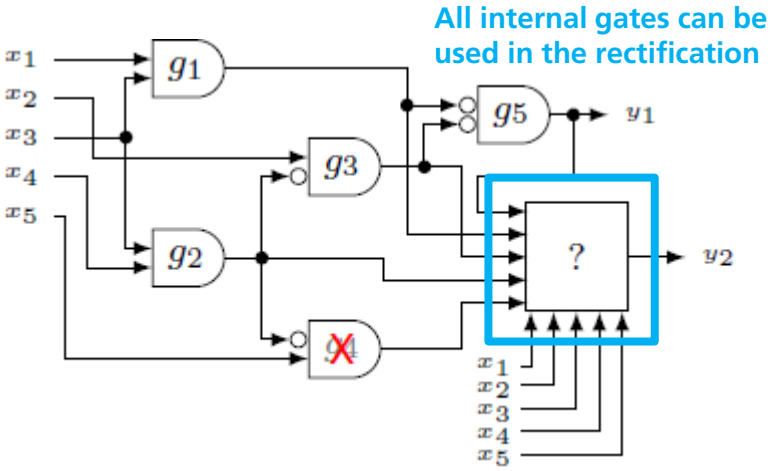
id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	

1. Counterexample computed with equivalence checking, e.g., ABC
2. Re-simulate counterexample to obtain outputs of internal gates

Patch: ?

Incremental patch synthesis by example





x "some" erroneous implementation

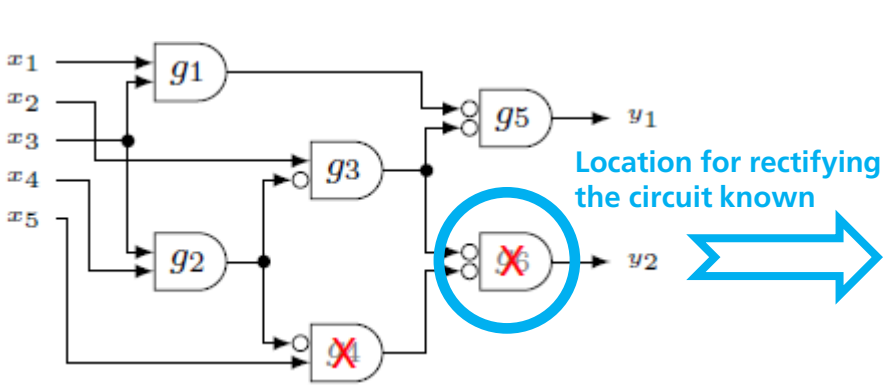
id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	

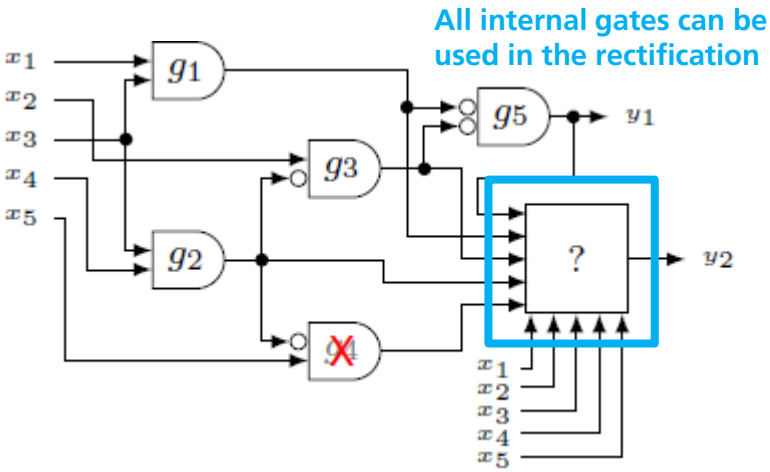
1. Counterexample computed with equivalence checking, e.g., ABC
2. Re-simulate counterexample to obtain outputs of internal gates

Internal gates are allowed too!

Patch: ?

Incremental patch synthesis by example





x "some" erroneous implementation

id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	

X_I X_R

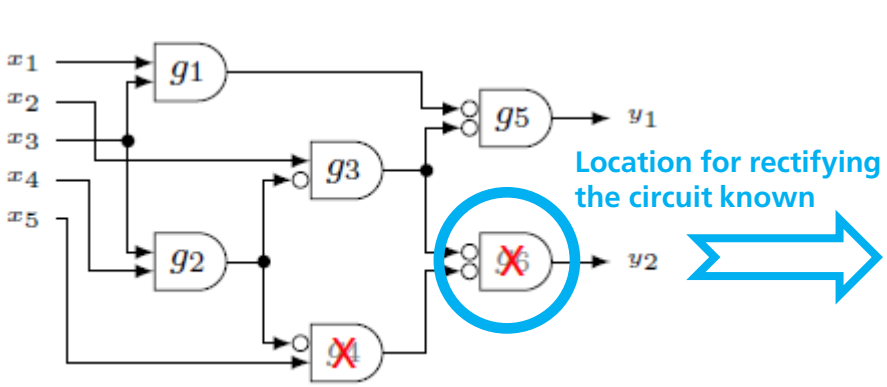
1. Counterexample computed with equivalence checking, e.g., ABC
2. Re-simulate counterexample to obtain outputs of internal gates

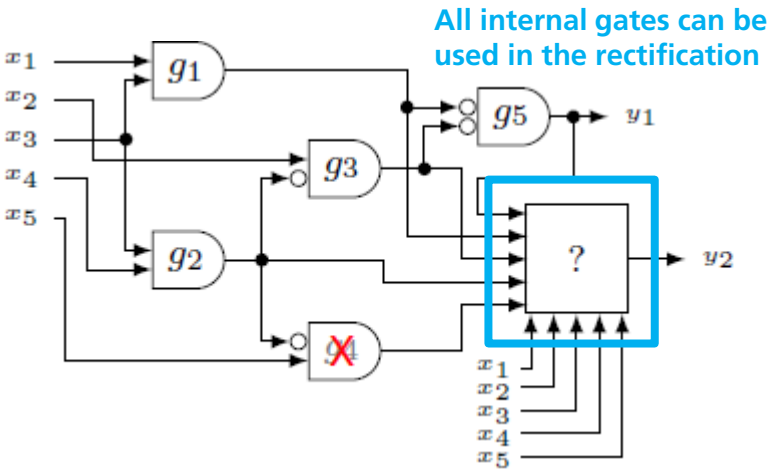
$$\exists F: \forall X_I: \exists X_R: Q(X_I, X_R, F(X_I, X_R))$$

Internal gates are allowed too!

Patch: ?

Incremental patch synthesis by example





x "some" erroneous implementation

id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	

X_I X_R

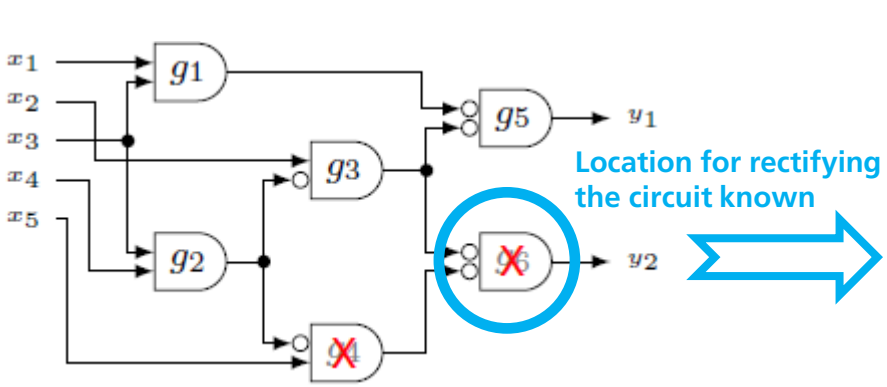
1. Counterexample computed with equivalence checking, e.g., ABC
2. Re-simulate counterexample to obtain outputs of internal gates

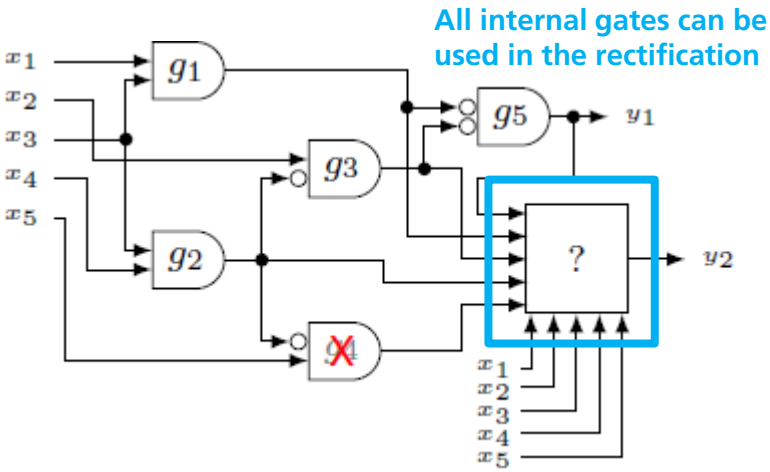
$$\exists F: \forall X_I: \exists X_R: Q(X_I, X_R, F(X_I, X_R))$$

Internal gates are allowed too!
(Another ∃-quantor, but totally bounded)

Patch: ?

Incremental patch synthesis by example





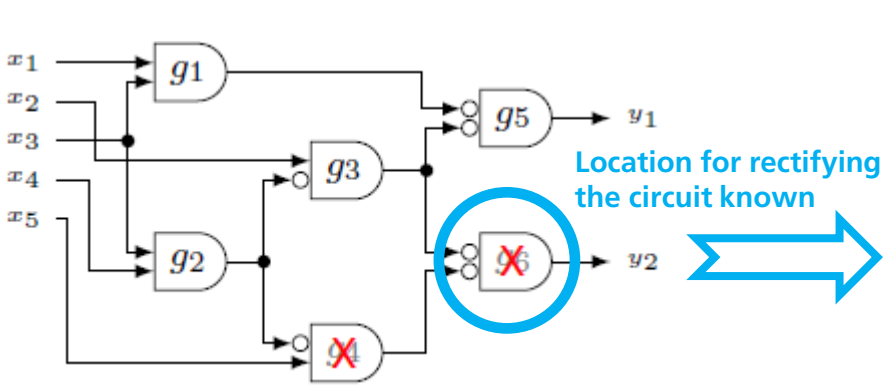
x "some" erroneous implementation

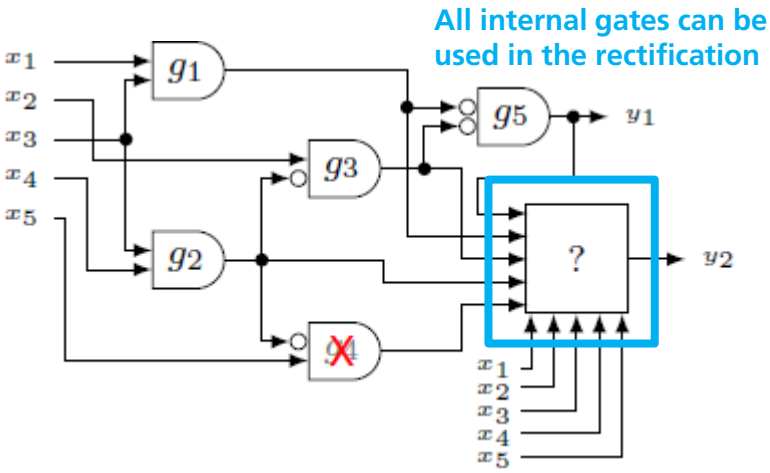
id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	

1. Counterexample computed with equivalence checking, e.g., ABC
2. Re-simulate counterexample to obtain outputs of internal gates

Patch: ?

Incremental patch synthesis by example





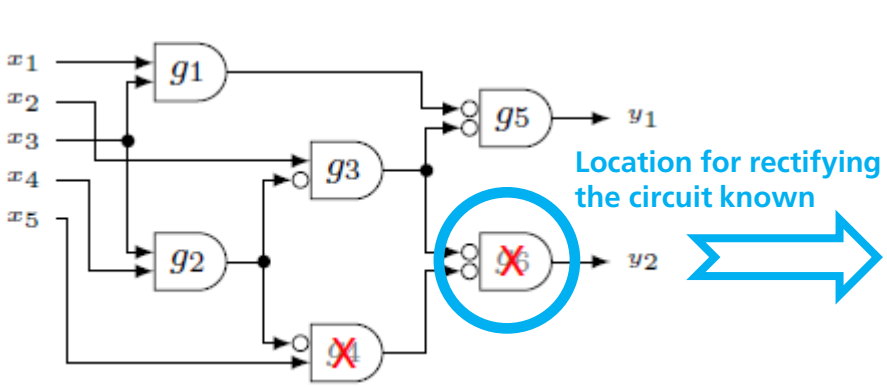
x "some" erroneous implementation

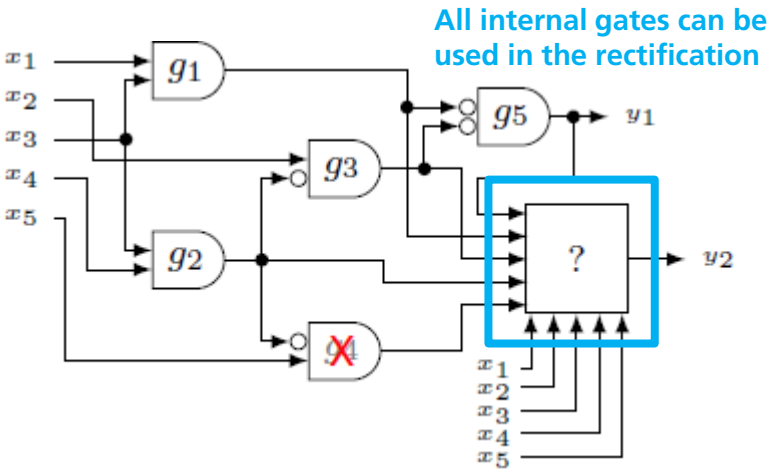
id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	1

1. Counterexample computed with equivalence checking, e.g., ABC
2. Re-simulate counterexample to obtain outputs of internal gates
3. Fix the correct value at g_6

Patch: ?

Incremental patch synthesis by example





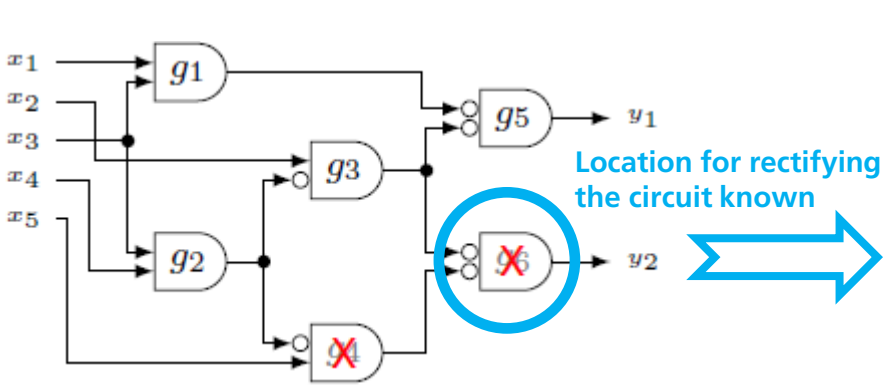
x "some" erroneous implementation

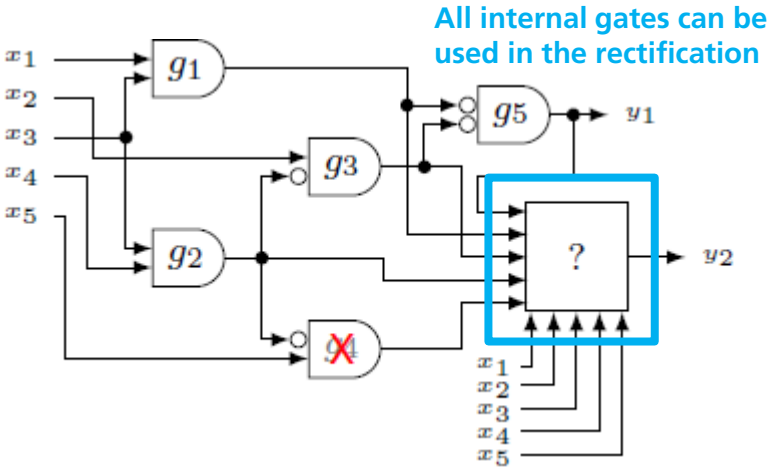
id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	1

1. Counterexample computed with equivalence checking, e.g., ABC
2. Re-simulate counterexample to obtain outputs of internal gates
3. Fix the correct value at g_6
4. Learn a circuit patch and insert it

Patch: $true$

Incremental patch synthesis by example





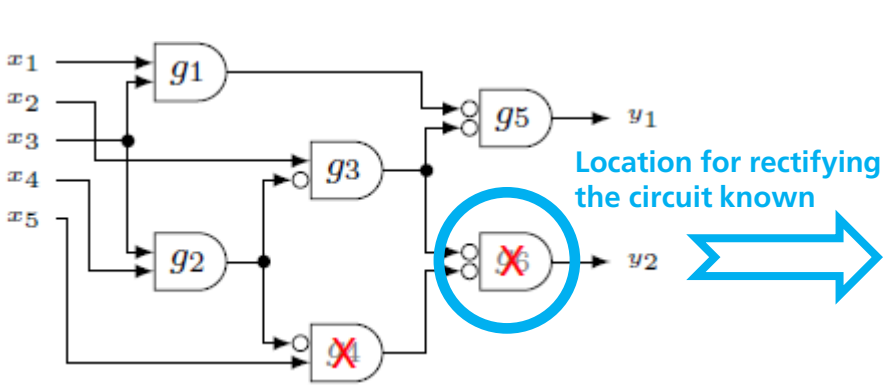
x "some" erroneous implementation

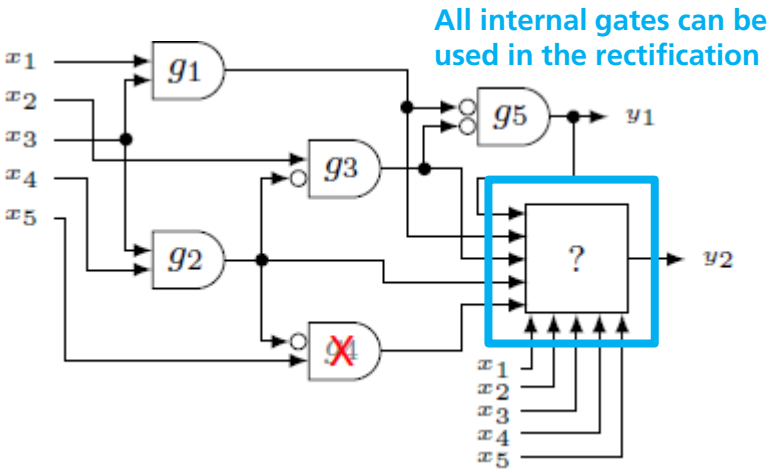
id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	1

1. Counterexample computed with equivalence checking, e.g., ABC
2. Re-simulate counterexample to obtain outputs of internal gates
3. Fix the correct value at g_6
4. Learn a circuit patch and insert it
5. Goto to step 1 to re-validate

Patch: $true$

Incremental patch synthesis by example





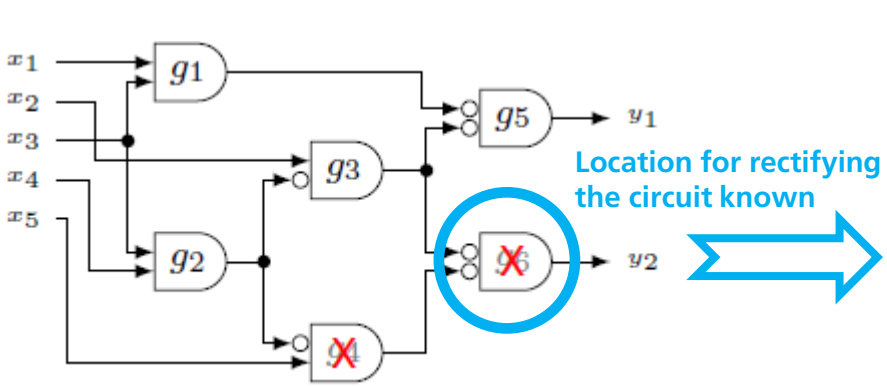
x "some" erroneous implementation

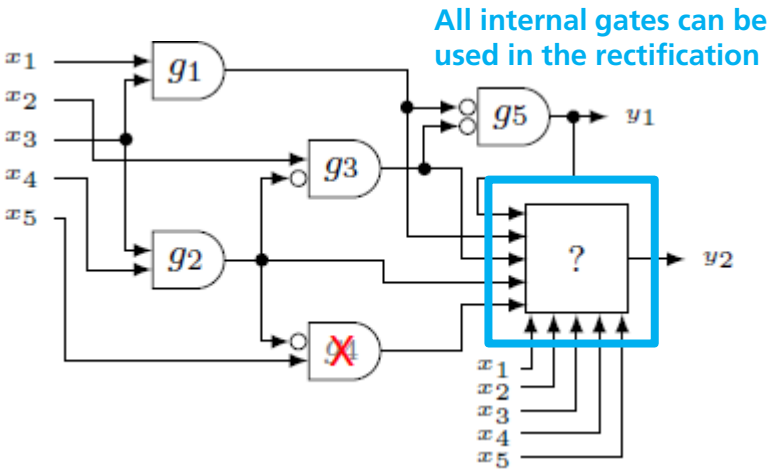
id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	1
2.	**1**	**1**	**0**	**0**	**1**	0	0	1	0	0	**1**	**0**

1. Counterexample computed with equivalence checking, e.g., ABC
2. Re-simulate counterexample to obtain outputs of internal gates
3. Fix the correct value at g_6
4. Learn a circuit patch and insert it
5. Goto to step 1 to re-validate

Patch: ~~*true*~~

Incremental patch synthesis by example





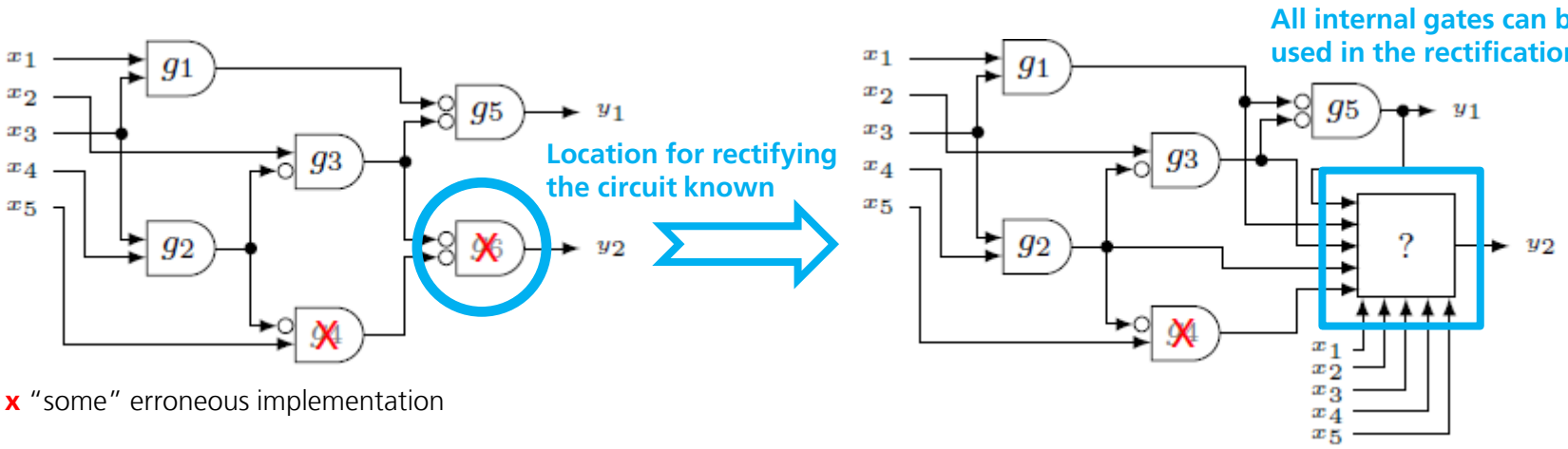
x "some" erroneous implementation

id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	1
2.	**1**	**1**	**0**	**0**	**1**	0	0	1	0	0	1	0

1. Counterexample computed with equivalence checking, e.g., ABC
2. Re-simulate counterexample to obtain outputs of internal gates
3. Fix the correct value at g_6
4. Learn a circuit patch and insert it
5. Goto to step 1 to re-validate

Patch: $(\neg x_1)$

Incremental patch synthesis by example



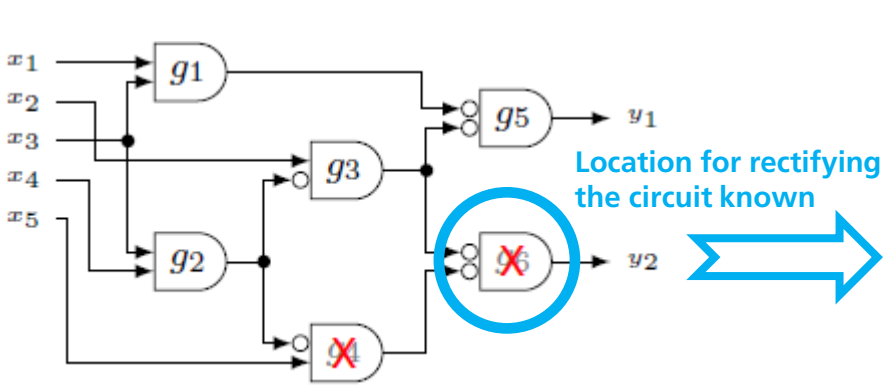
x "some" erroneous implementation

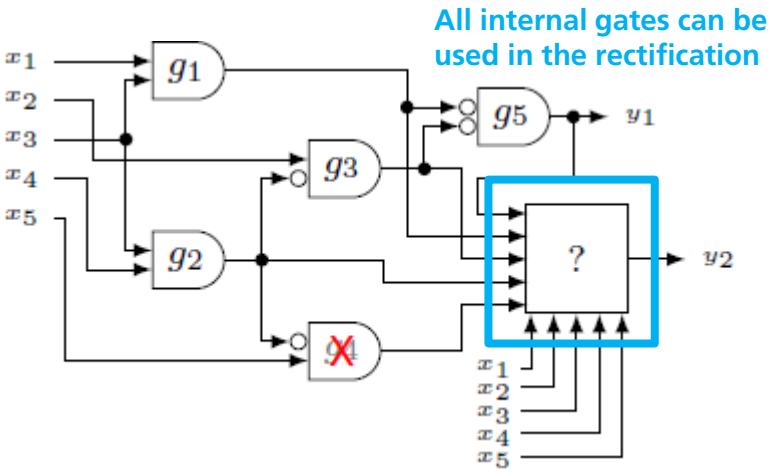
id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	1
2.	1	1	0	0	1	0	0	1	0	0	1	0
3.	**1**	**1**	**1**	**1**	**1**	1	1	0	1	0	0	1

1. Counterexample computed with equivalence checking, e.g., ABC
2. Re-simulate counterexample to obtain outputs of internal gates
3. Fix the correct value at g_6
4. Learn a circuit patch and insert it
5. Goto to step 1 to re-validate

Patch: $(\neg x_1)$

Incremental patch synthesis by example





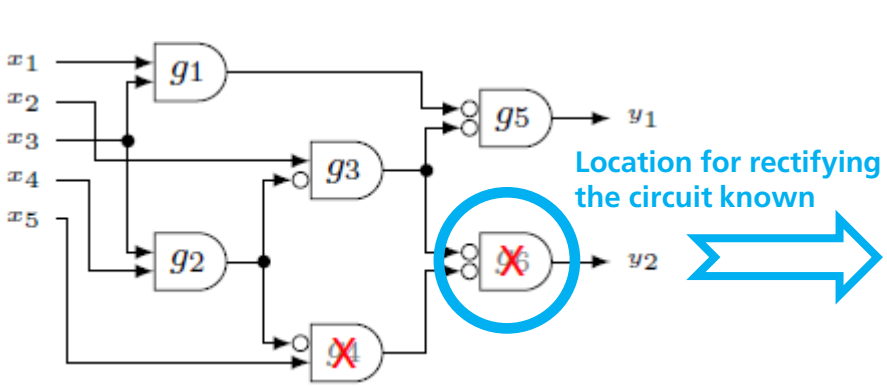
x "some" erroneous implementation

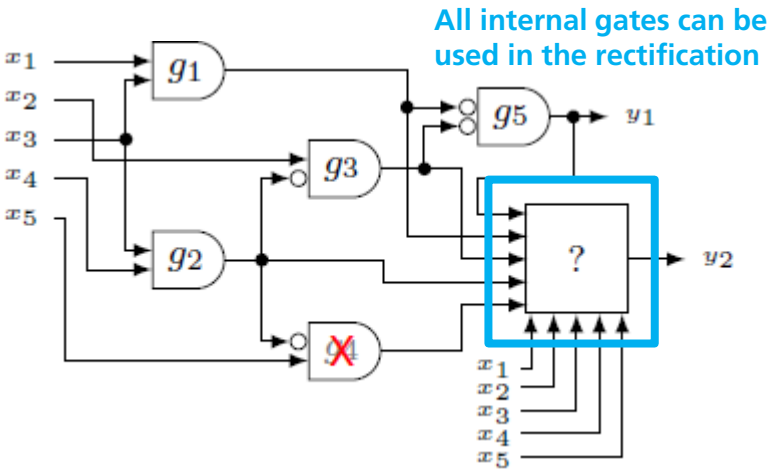
id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	1
2.	1	1	0	0	1	0	0	1	0	0	1	0
3.	**1**	**1**	**1**	**1**	**1**	1	1	0	1	0	0	1

1. Counterexample computed with equivalence checking, e.g., ABC
2. Re-simulate counterexample to obtain outputs of internal gates
3. Fix the correct value at g_6
4. Learn a circuit patch and insert it
5. Goto to step 1 to re-validate

Patch: $(\neg g_3)$

Incremental patch synthesis by example





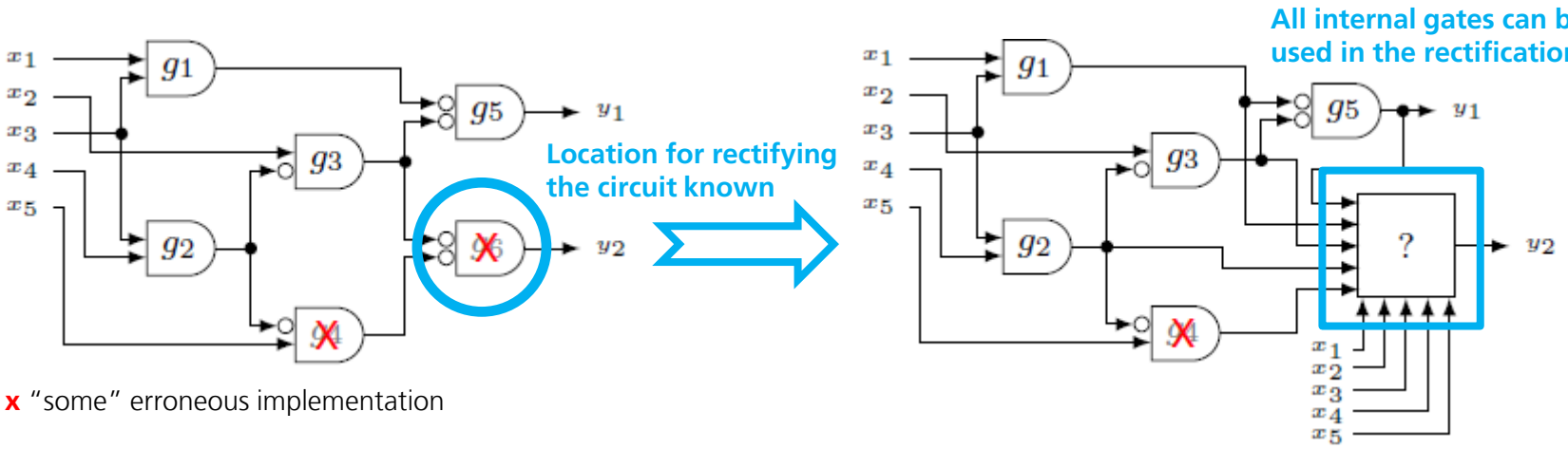
x "some" erroneous implementation

id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	1
2.	1	1	0	0	1	0	0	1	0	0	1	0
3.	1	1	1	1	1	1	1	0	1	0	0	1
4.	**0**	**0**	**0**	**1**	**1**	0	0	0	0	1	1	0

Patch: $(\neg g_3)$

1. Counterexample computed with equivalence checking, e.g., ABC
2. Re-simulate counterexample to obtain outputs of internal gates
3. Fix the correct value at g_6
4. Learn a circuit patch and insert it
5. Goto to step 1 to re-validate

Incremental patch synthesis by example



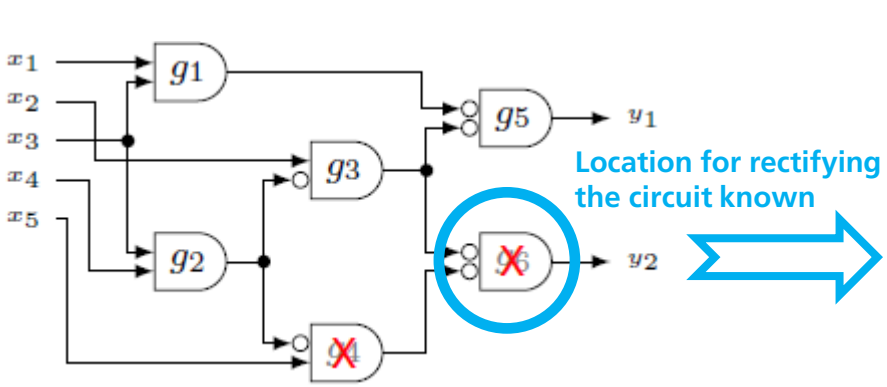
x "some" erroneous implementation

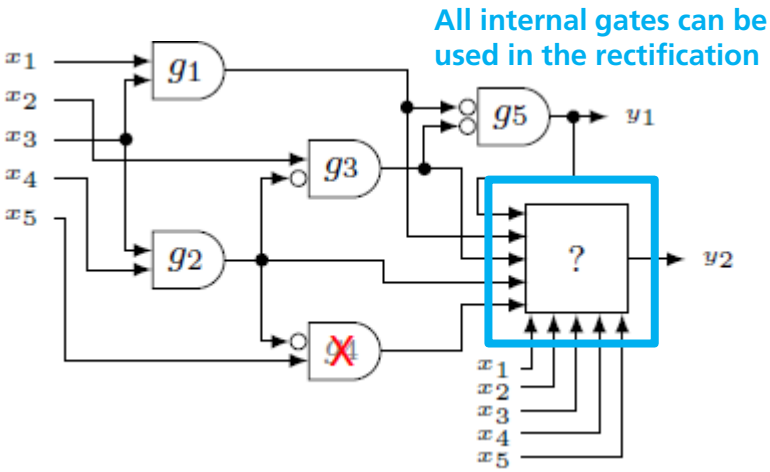
id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	1
2.	1	1	0	0	1	0	0	1	0	0	1	0
3.	1	1	1	1	1	1	1	0	1	0	0	1
4.	**0**	**0**	**0**	**1**	**1**	0	0	0	0	1	1	0

Patch: $(\neg x_4 \wedge \neg g_3) \vee (g_2)$

1. Counterexample computed with equivalence checking, e.g., ABC
2. Re-simulate counterexample to obtain outputs of internal gates
3. Fix the correct value at g_6
4. Learn a circuit patch and insert it
5. Goto to step 1 to re-validate

Incremental patch synthesis by example





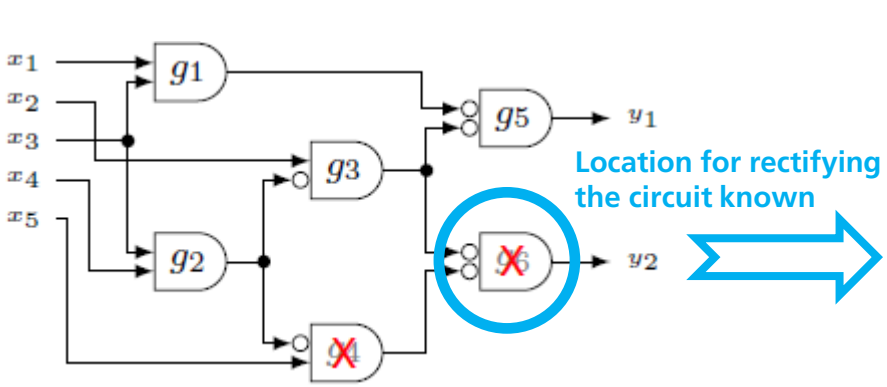
x "some" erroneous implementation

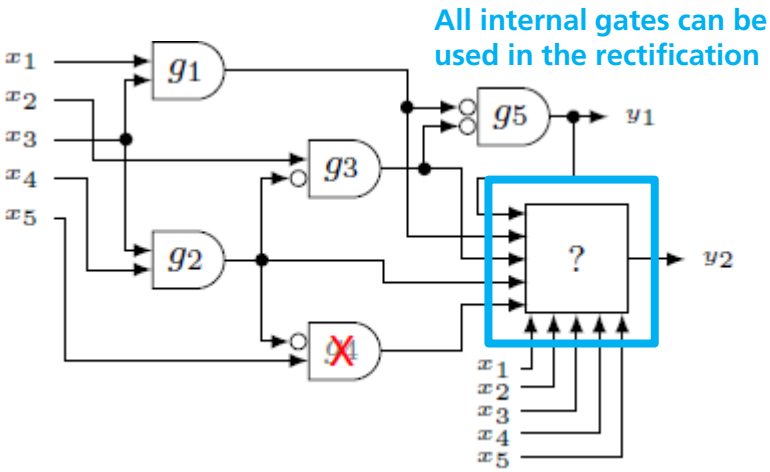
id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	1
2.	1	1	0	0	1	0	0	1	0	0	1	0
3.	1	1	1	1	1	1	1	0	1	0	0	1
4.	0	0	0	1	1	0	0	0	0	1	1	0
5.	**0**	**0**	**0**	**1**	**0**	0	0	0	0	1	0	1

Patch: $(\neg x_4 \wedge \neg g_3) \vee (g_2)$

1. Counterexample computed with equivalence checking, e.g., ABC
2. Re-simulate counterexample to obtain outputs of internal gates
3. Fix the correct value at g_6
4. Learn a circuit patch and insert it
5. Goto to step 1 to re-validate

Incremental patch synthesis by example





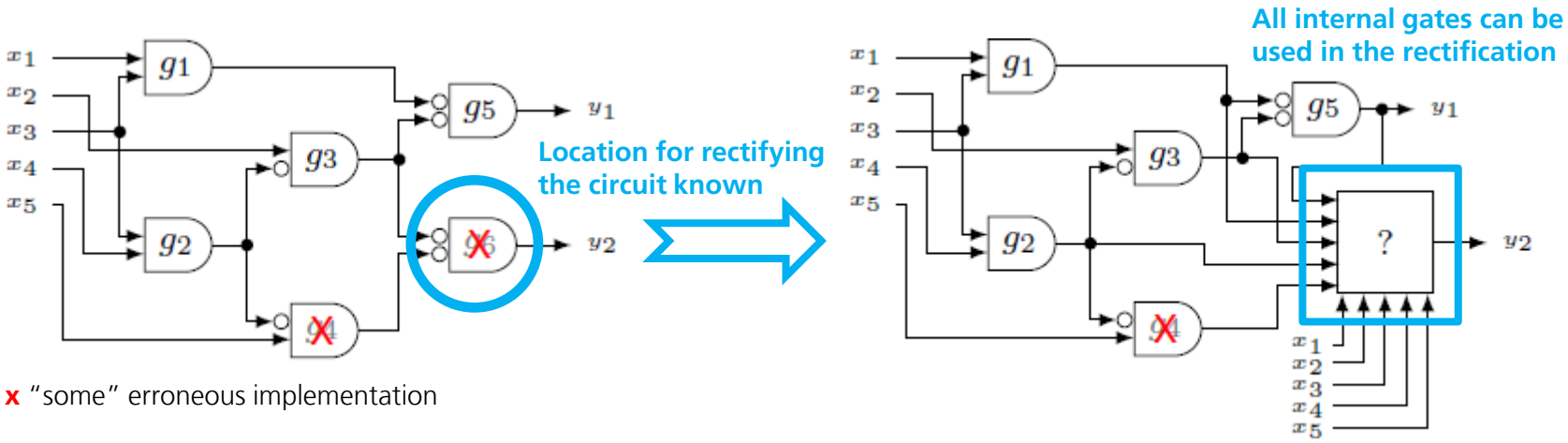
x "some" erroneous implementation

id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	1
2.	1	1	0	0	1	0	0	1	0	0	1	0
3.	1	1	1	1	1	1	1	0	1	0	0	1
4.	0	0	0	1	1	0	0	0	0	1	1	0
5.	**0**	**0**	**0**	**1**	**0**	0	0	0	0	1	0	1

Patch: $(\neg x_5 \wedge \neg g_3) \vee (g_2)$

1. Counterexample computed with equivalence checking, e.g., ABC
2. Re-simulate counterexample to obtain outputs of internal gates
3. Fix the correct value at g_6
4. Learn a circuit patch and insert it
5. Goto to step 1 to re-validate

Incremental patch synthesis by example

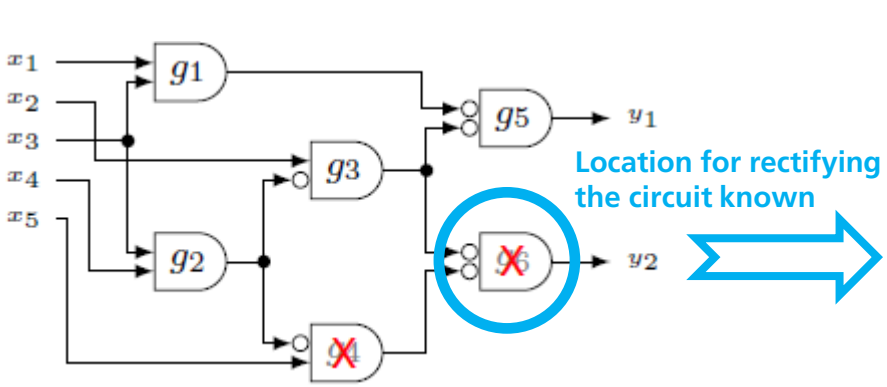


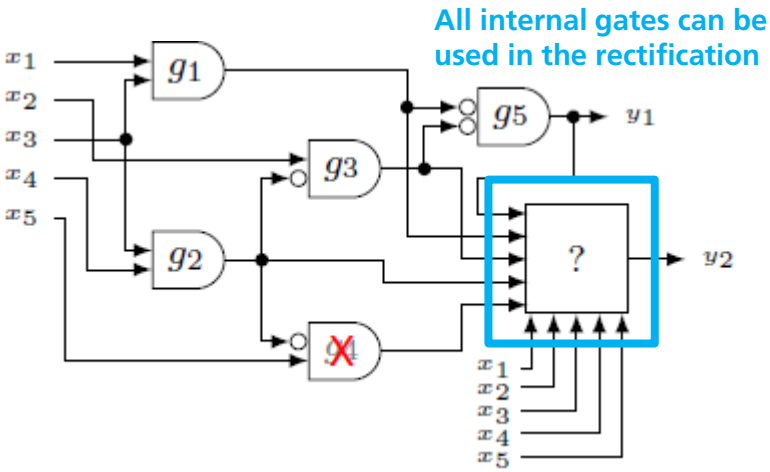
x "some" erroneous implementation

id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	**0**	**1**
2.	1	1	0	0	1	0	0	1	0	0	**1**	**0**
3.	1	1	1	1	1	1	1	0	1	0	**0**	**1**
4.	0	0	0	1	1	0	0	0	0	1	**1**	**0**
5.	0	0	0	1	0	0	0	0	0	1	**0**	**1**

Patch: $(\neg x_5 \wedge \neg g_3) \vee (g_2)$ **CORRECT!** (No new counterexamples)

Incremental patch synthesis by example





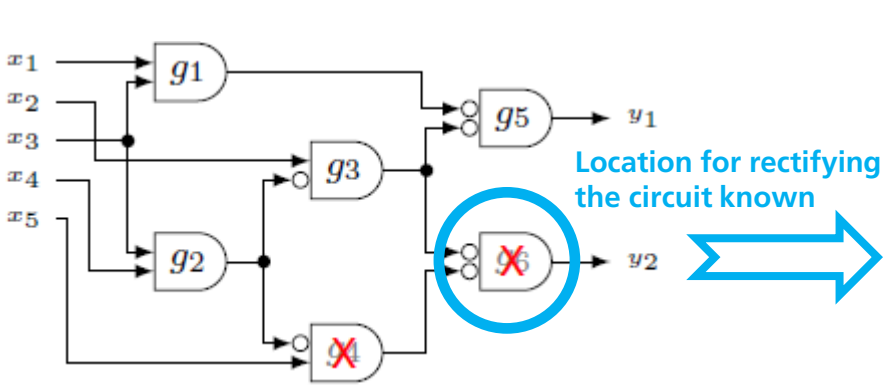
x "some" erroneous implementation

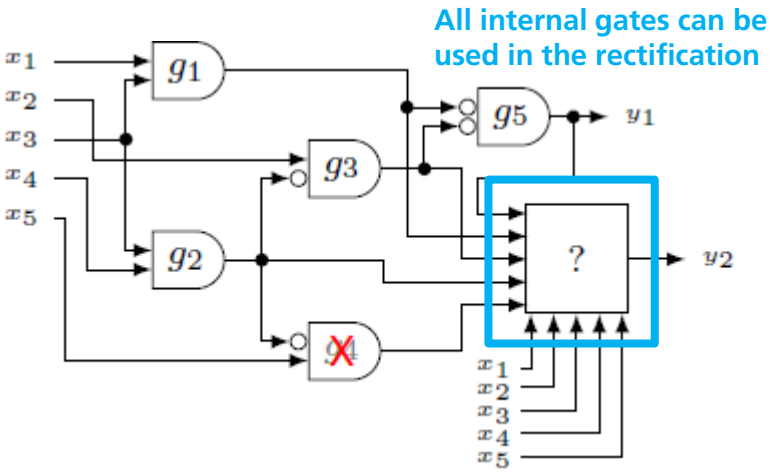
id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	1
2.	1	1	0	0	1	0	0	1	0	0	1	0
3.	1	1	1	1	1	1	1	0	1	0	0	1
4.	0	0	0	1	1	0	0	0	0	1	1	0
5.	0	0	0	1	0	0	0	0	0	1	0	1

- 5 samples (16%) suffice for re-synthesis
- Only single-output functions considered (correct output g_6 obtained by negation)
- Multi-output functions more possibilities have to be considered (computational more intensive!)
- No optimization of final patch

Patch: $(\neg x_5 \wedge \neg g_3) \vee (g_2)$ **CORRECT!** (No new counterexamples)

Incremental patch synthesis by example





x "some" erroneous implementation

id	x_1	x_2	x_3	x_4	x_5	g_1	g_2	g_3	g_4	g_5	g_6	g_6
1.	0	0	0	0	0	0	0	0	0	1	0	1
2.	1	1	0	0	1	0	0	1	0	0	1	0
3.	1	1	1	1	1	1	1	0	1	0	0	1
4.	0	0	0	1	1	0	0	0	0	1	1	0
5.	0	0	0	1	0	0	0	0	0	1	0	1

- 5 samples (16%) suffice for re-synthesis
- Only single-output functions considered (correct output g_6 obtained by negation)
- Multi-output functions more possibilities have to be considered (computational more intensive!)
- No optimization of final patch

Patch: $(\neg x_5 \wedge \neg g_3) \vee (g_2)$ **CORRECT!** (No new counterexamples)

Experimental setting

- Implemented the approach in C++ and evaluated it for circuit rectification
  - Satisfiability problems are solved using MiniSAT
  - Counterexamples are computed using the combinational equivalence checker of ABC and are re-simulated to obtain assignment for all internal gates
- Experiments conducted on a quad-core Intel i5-2520M CPU with 2.50GHz and 8GB RAM running Linux kernel 4.5.4-1

Rectifying multiple faults at a single location may be difficult

- No standard benchmark set for rectification available
- Seeded multiple faults into gate level circuits: ISCAS + EPFL benchmarks
- **Side-constraint:** the seeded faults can be rectified at a single location and the location is known (we used our **exact fault localization [ICCAD16]**)
  - ISCAS/EPFL benchmarks without this side-constraint were not considered
- 10 faulty versions of each circuit

Experimental evaluation: Restricted to SOP up to 2 cubes

Name	R	U	T/O	N	I	t_R	t_U	t_Σ
c432	6	4	0	175.9	50.50	0.14	0.15	0.1
cavlc	0	10	0	702.0	103.6	0.03	0.00	0.1
int2float	0	9	1	270.0	60.0	0.00	65.93	83.0
priority	0	10	0	1105.0	120.8	0.00	5.53	5.5
router	1	9	0	293.9	45.7	0.50	0.58	0.6

R … rectified
U … unrectifiable with 2-SOP (proven)
T/O … timeout

N … mean number of variables
I … mean number of iterations

t_R … time for rectified circuit (mean)
t_U … time for unrectifiable circuits (mean)
t_Σ … time for rectified or unrectifiable circuits (mean)

Time limit: 100s per benchmark

Experimental evaluation: Restricted to SOP up to 3 cubes

Name	R	U	T/O	N	I	t_R	t_U	t_Σ
c432	7	3	0	175.9	72.8	0.31	38.90	11.9
cavlc	3	1	6	702.0	232.2	3.14	96.89	70.6
int2float	5	4	1	270.0	141.6	1.12	20.11	8.6
priority	1	1	8	1105.0	237.7	5.58	19.40	82.5
router	2	7	1	293.9	84.9	0.62	3.33	12.5

R … rectified
U … unrectifiable with 3-SOP (proven)
T/O … timeout

N … mean number of variables
I … mean number of iterations

t_R … time for rectified circuit (mean)
t_U … time for unrectifiable circuits (mean)
t_Σ … time for rectified or unrectifiable circuits (mean)

Time limit: 100s per benchmark

Summary & conclusion

- Extension of the EF-synthesis problem by allowing existential quantification over Boolean functions with an application to circuit rectification or ECO-synthesis

- A CEGAR-based approach for determining Boolean function realizations in normal form representation (of bounded size)
  - SAT-based bound synthesis = Comb. equiv. checking + Boolean learning

- Experimental results for circuit rectification of *some* ISCAS and EPFL benchmarks
  - Multiple seeded faults are corrected at a single location
  - Sum-Of-Products representation

CEGAR-based EF Synthesis of Boolean Functions with an Application to Circuit Rectification

Heinz Riener, Rüdiger Ehlers, and Goerschwin Fey
German Aerospace Center, Bremen, Germany
DFKI GmbH, Germany
University of Bremen, Germany

DLR