Efficient representations of digital circuits in EDA-Applications

Prof. Dr. René Krenz-Baath

Technische Hochschule Wildau
Technical University of Applied Sciences
WILDAU

IDEAS - Systems - Innovation - Development - Engineering - Additive Manufacturing -

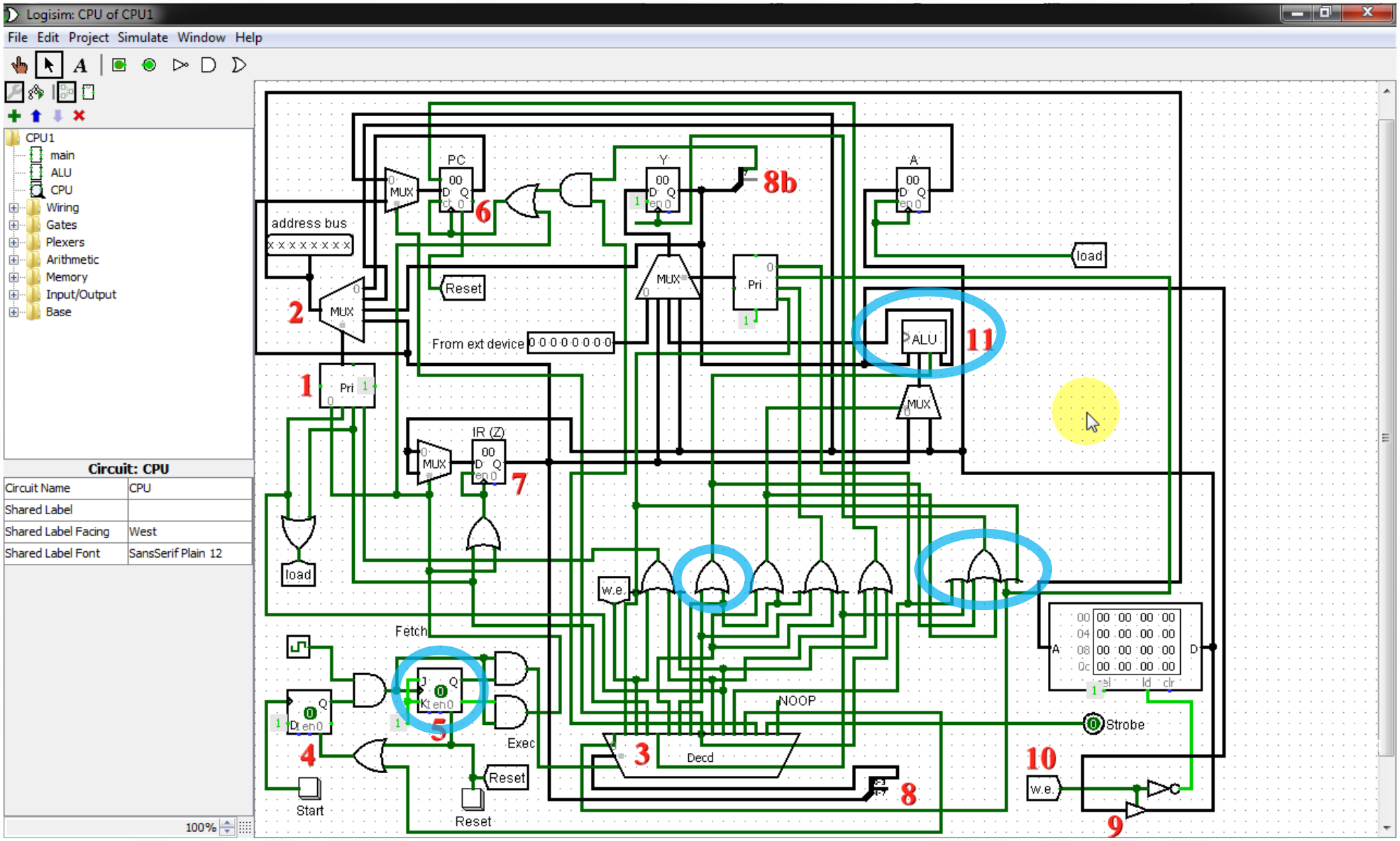
Overview

- introduction to data structures in EDA (circuit graphs AIGs, FSMs, BDDs, SAT (Watchers))
  + representations on different abstraction levels
  + normal circuit graphs
  + AIGs
  + FSMs
  + BDDs
- multi-valued CNF-structures / miniSAT
  + introduction MV-ATPG
  + MV-CNF-Encoding
  + MiniSat CNF-treatment
- BDD-based CNF-reduction in FFRs
  + SAT-ATPG CNF-Encoding
  + FFRs-based CNF-generation
  + BDD2CNF; HowTo; rebirth rates; BDD-Varaible-sharing
- BDD-variable recycling
  + motivation
  + basic idea
  + experimental results
- Conclusions

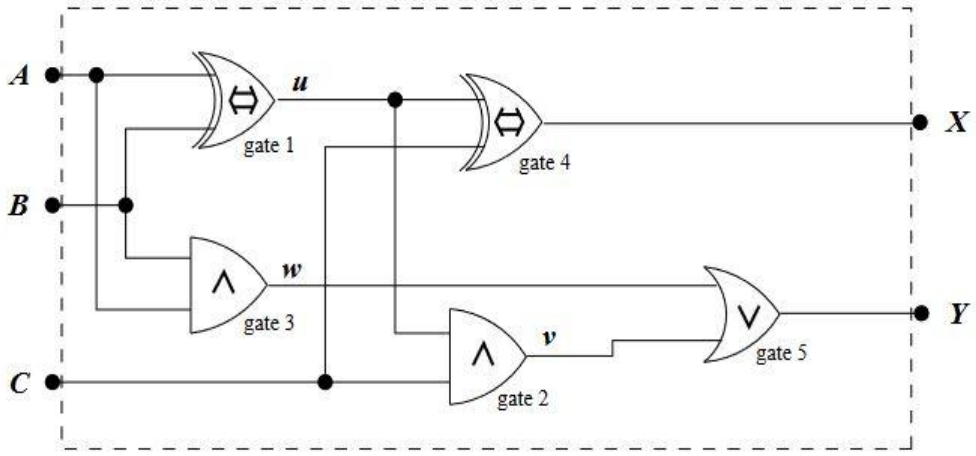
Efficient representations of digital circuits in EDA-Applications

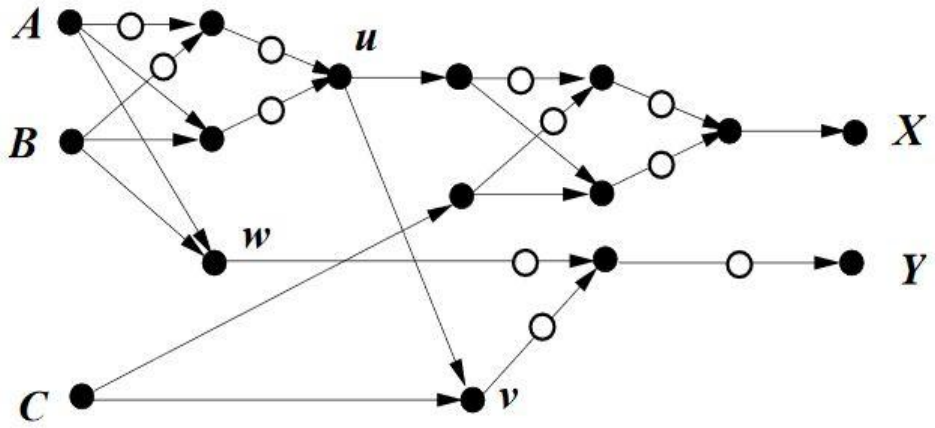
- Perspective of EDA-Software Architect
- data structures in EDA-Tools
- data structures vs algorithms

Introduction to data structures in EDA



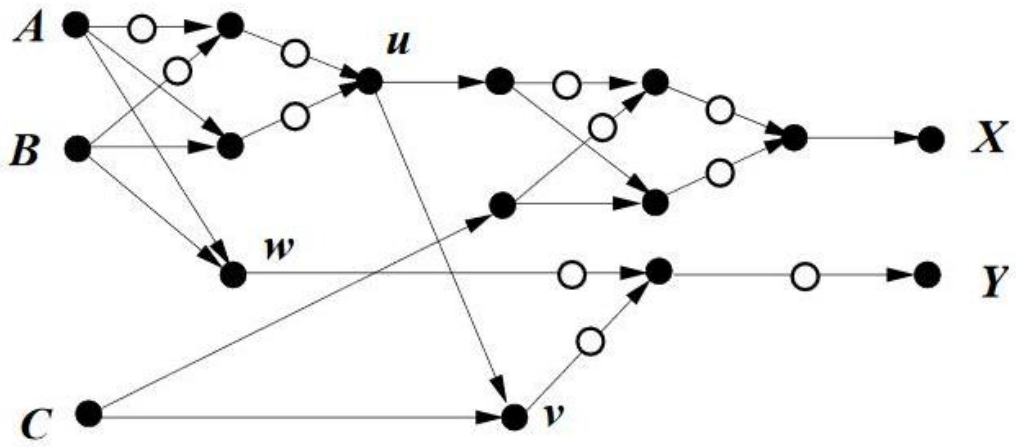
Introduction to data structures in EDA - AIGs





- Representing combinational digital circuits
- specific type of **Directed Acyclic Graph (DAG)**
- Consisting of primary inputs, AND-vertices, constant vertices, primary outputs
- Complemented edges represent inverters
- AND-gates and inverters represent a **functionally complete set**
- Homogeneous Structure: uniform and simple structure support efficient reasoning algorithms
- **Non-canonical** representation of a circuit
- Widely applied in logic synthesis, formal verification, CEC

Introduction to data structures in EDA - AIGs

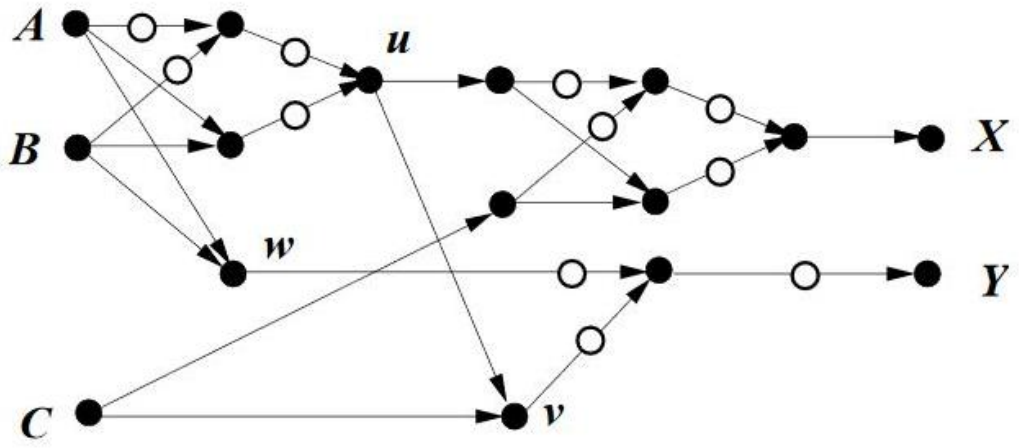


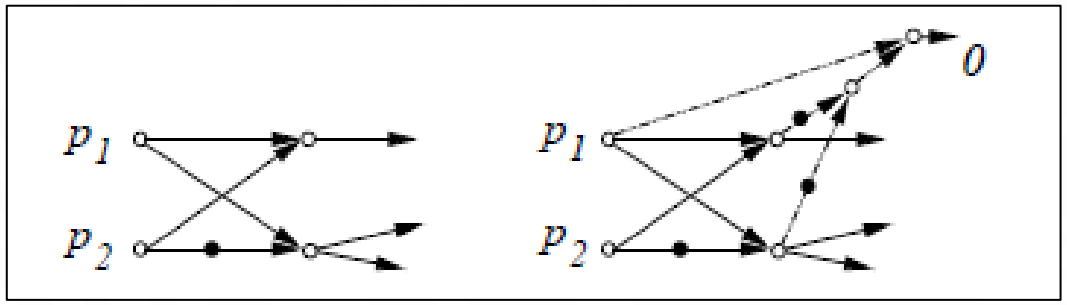

```
Algorithm new_and_vertex(p1, p2)
    p = alloc_vertex(p1, p2);
    add_to_hash_table(p, p1,p2);

    if ((b1 = hash_lookup(¬p1, p2)) != NULL) learn(p,b1);
    if ((b2 = hash_lookup(p1, ¬ p2)) != NULL) learn(p,b2);

    /* reschedule for BDD sweeping */
    put_on_heap( bdd_from_vertex(p1), upper size limit);
    put_on_heap( bdd_from_vertex(p2), upper size limit);
    return p;
```

```
Algorithm and(p1, p2) /* constant foldin g */
    if (p1 == CONST0) return CONST0;
    if (p2 == CONST0) return CONST0;
    if (p1 == CONST1) return p2;
    if (p2 == CONST1) return p1;
    if (p1 == p2) return p1;
    if (p1 == p2) return CONST0;

    /* rank order inputs */
    if (rank(p1) > rank(p2)) swap(p1, p2);

    /* check for isomorphic entry in hash table */
    if (p = hash_lookup(p1, p2)) return p;

    /* 3 cases depending on position in circuit */
    if (is_var(p1) && is_var(p2))
        return new_and_vertex(p1,p2);

    else if (is var(p1)) return and3(p1,p2);
    else if (is var(p2)) return and3(p2,p1);
    else return and4(p1,p2);
```

A. Kuehlmann and F. Krohm, "Equivalence Checking Using Cuts And Heaps," *Proc. DAC'97*

Introduction to data structures in EDA – Circuit-based SAT-Solving



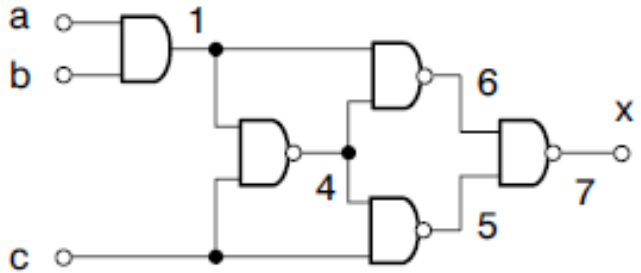


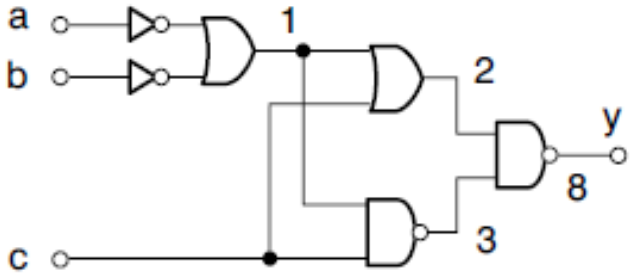
Current	Next	Action
1 X → X	1 X → X	STOP
0 X → 1	0 X → 1	CONFLICT
X X → 0	X X → 0	CASE_SPLIT
0 X → X	0 X → 0	PROP_FORWARD
X X → 1	1 1 → 1	PROP_LEFT_RIGHT
. . .	. . .	. . .

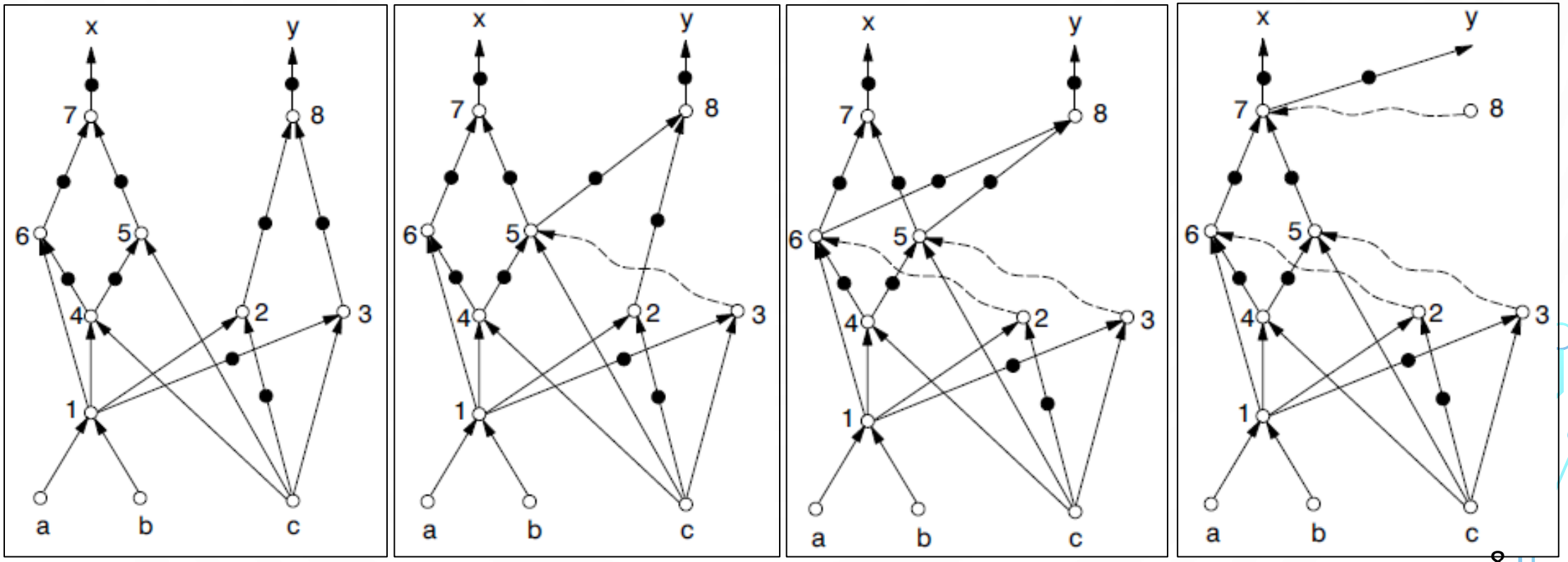
Only 27 (3^3) possible implication cases

A. Kuehlmann and F. Krohm, "Equivalence Checking Using Cuts And Heaps," *Proc. DAC'97*

Introduction to data structures in EDA – AIGs
Structural Hashing



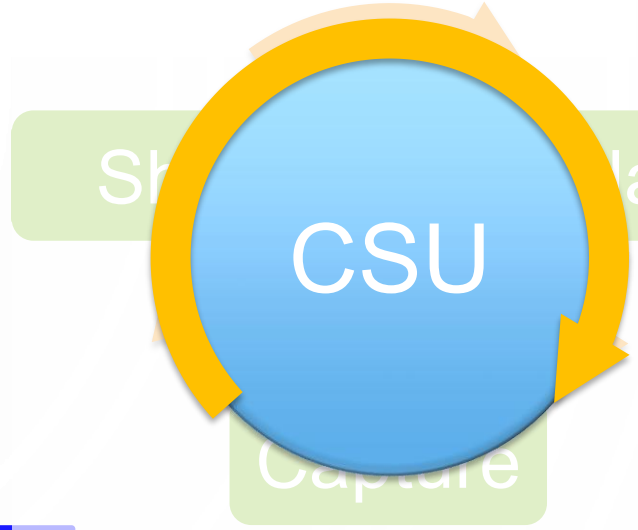


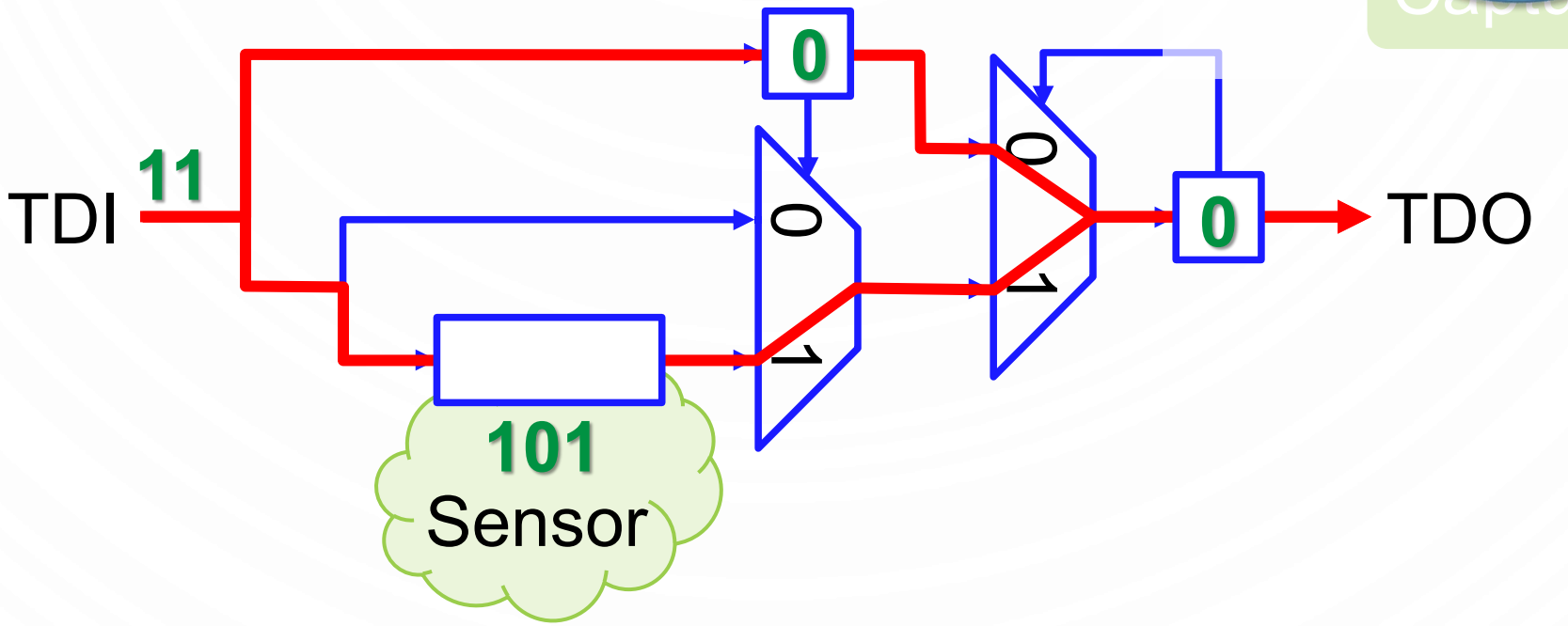


A. Kuehlmann and F. Krohm, "Equivalence Checking Using Cuts And Heaps," *Proc. DAC'97*

Introduction to data structures in EDA - FSMs

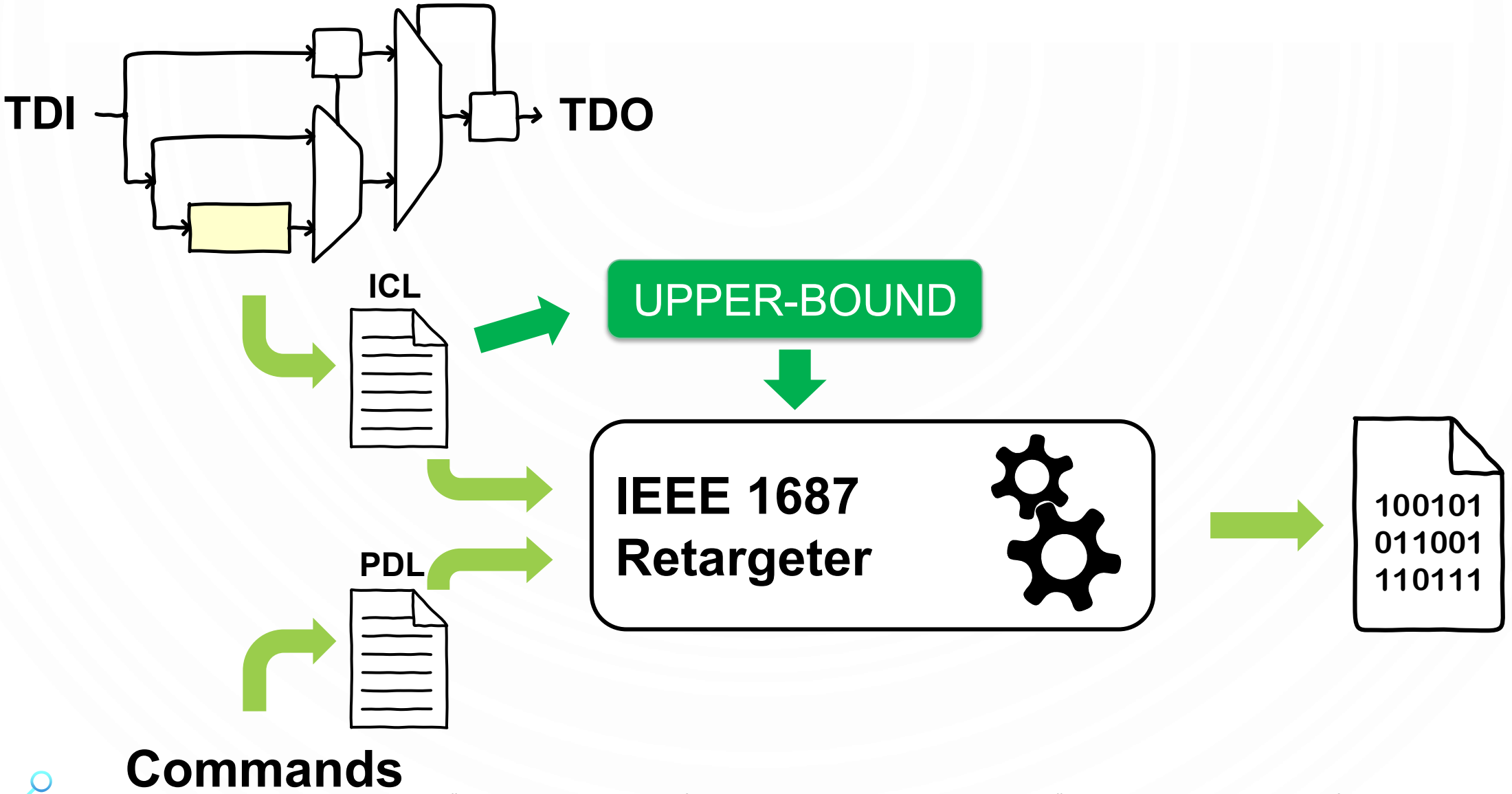
Accessing a sensor within an IJTAG-network





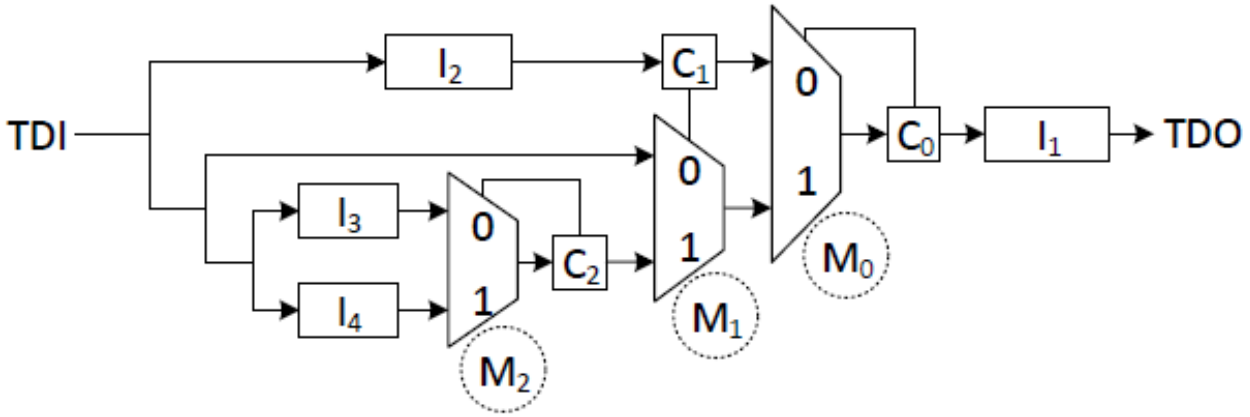
F. G. Zadegan, R. Krenz-Baath and E. Larsson, "Upper-bound computation for optimal retargeting in IEEE1687 networks," *2016 IEEE International Test Conference (ITC)*, 2016

Introduction to data structures in EDA - FSMs

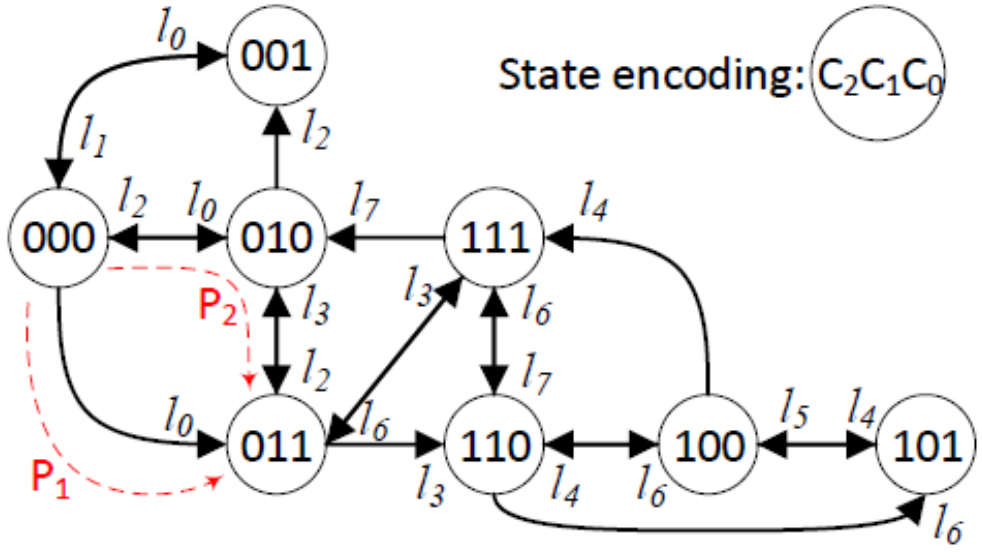


F. G. Zadegan, R. Krenz-Baath and E. Larsson, "Upper-bound computation for optimal retargeting in IEEE1687 networks," *2016 IEEE International Test Conference (ITC)*, 2016

Introduction to data structures in EDA - FSMs

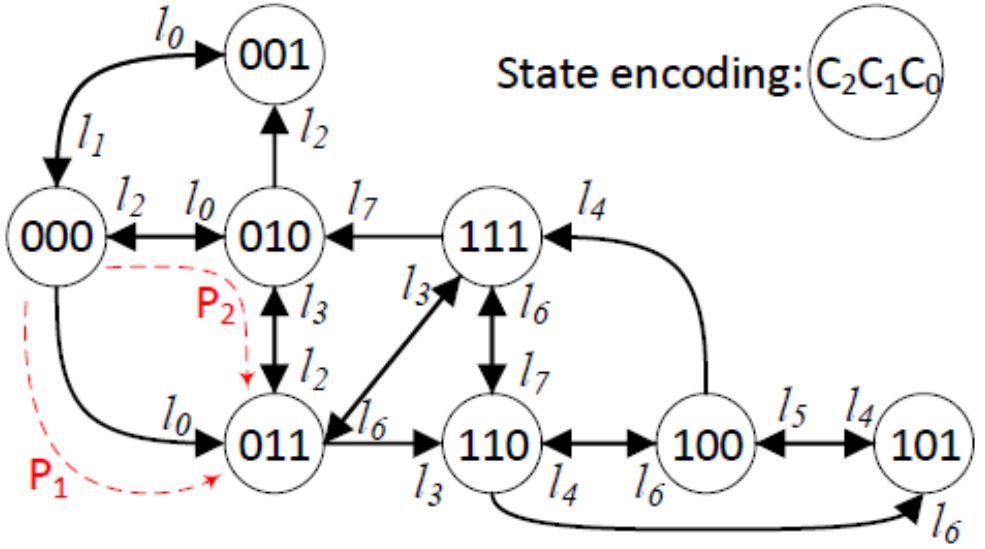


State	Active components
000	I_2, C_1, C_0, I_1
001	C_0, I_1
010	I_2, C_1, C_0, I_1
011	I_3, C_2, C_0, I_1
100	I_2, C_1, C_0, I_1
101	C_0, I_1
110	I_2, C_1, C_0, I_1
111	I_4, C_2, C_0, I_1



F. G. Zadegan, R. Krenz-Baath and E. Larsson, "Upper-bound computation for optimal retargeting in IEEE1687 networks," *2016 IEEE International Test Conference (ITC)*, 2016

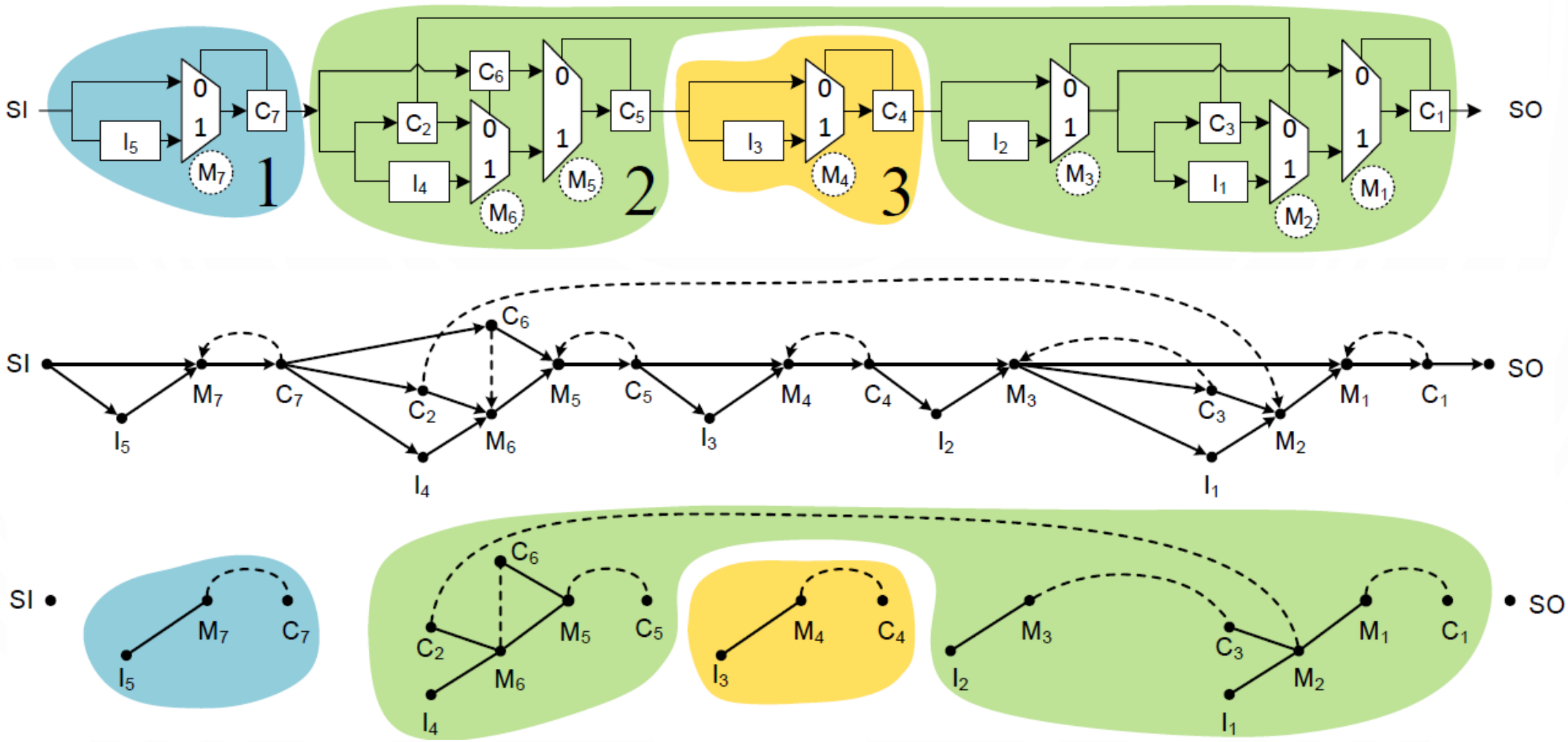
Introduction to data structures in EDA - FSMs



State	000	001	010	011	100	101	110	111
000	0	1	1	1	3	3	2	2
001	1	0	2	2	4	4	3	3
010	1	1	0	1	3	3	2	2
011	2	2	1	0	2	2	1	1
100	3	3	2	2	0	1	1	1
101	4	4	3	3	1	0	2	2
110	3	3	2	2	1	1	0	1
111	2	2	1	1	2	2	1	0

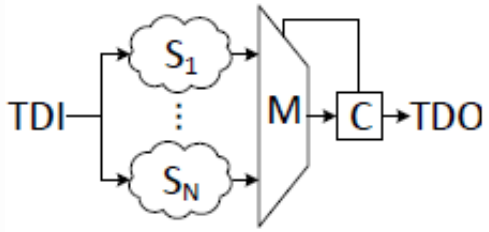
F. G. Zadegan, R. Krenz-Baath and E. Larsson, "Upper-bound computation for optimal retargeting in IEEE1687 networks," *2016 IEEE International Test Conference (ITC)*, 2016

Introduction to data structures in EDA – FSMs Decomposition

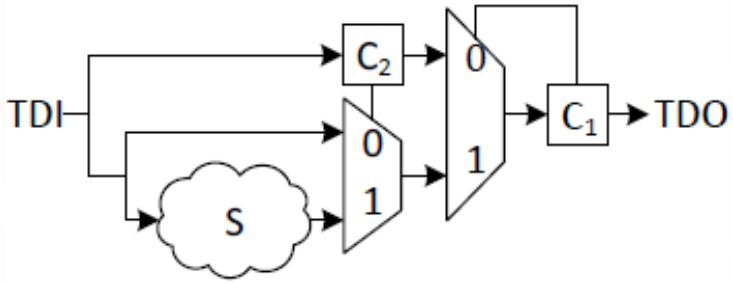


F. G. Zadegan, R. Krenz-Baath and E. Larsson, "Upper-bound computation for optimal retargeting in IEEE1687 networks," *2016 IEEE International Test Conference (ITC)*, 2016

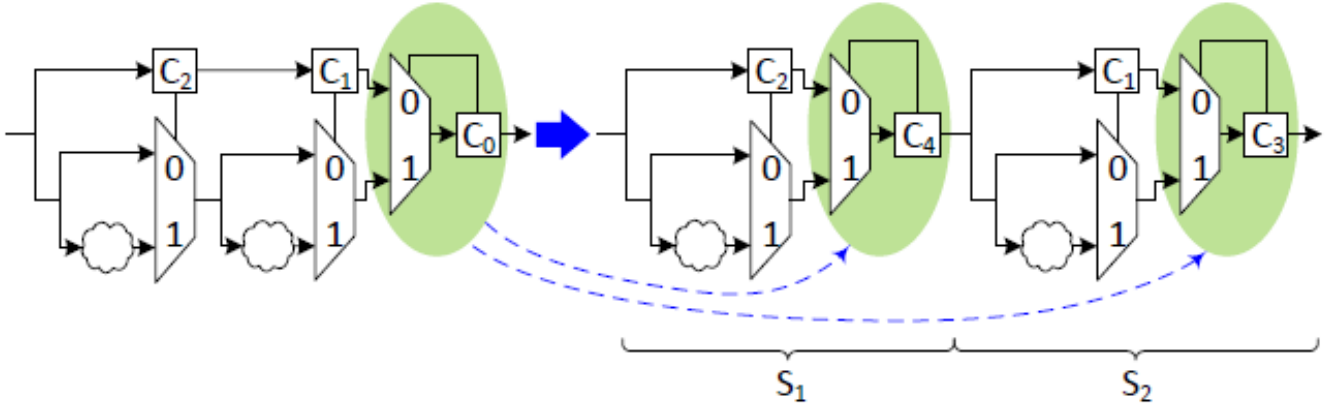
Introduction to data structures in EDA – FSMs
Decompose, Lookup, Rewrite



Decompose:
Max_UCC(S_1,…,S_N) +1



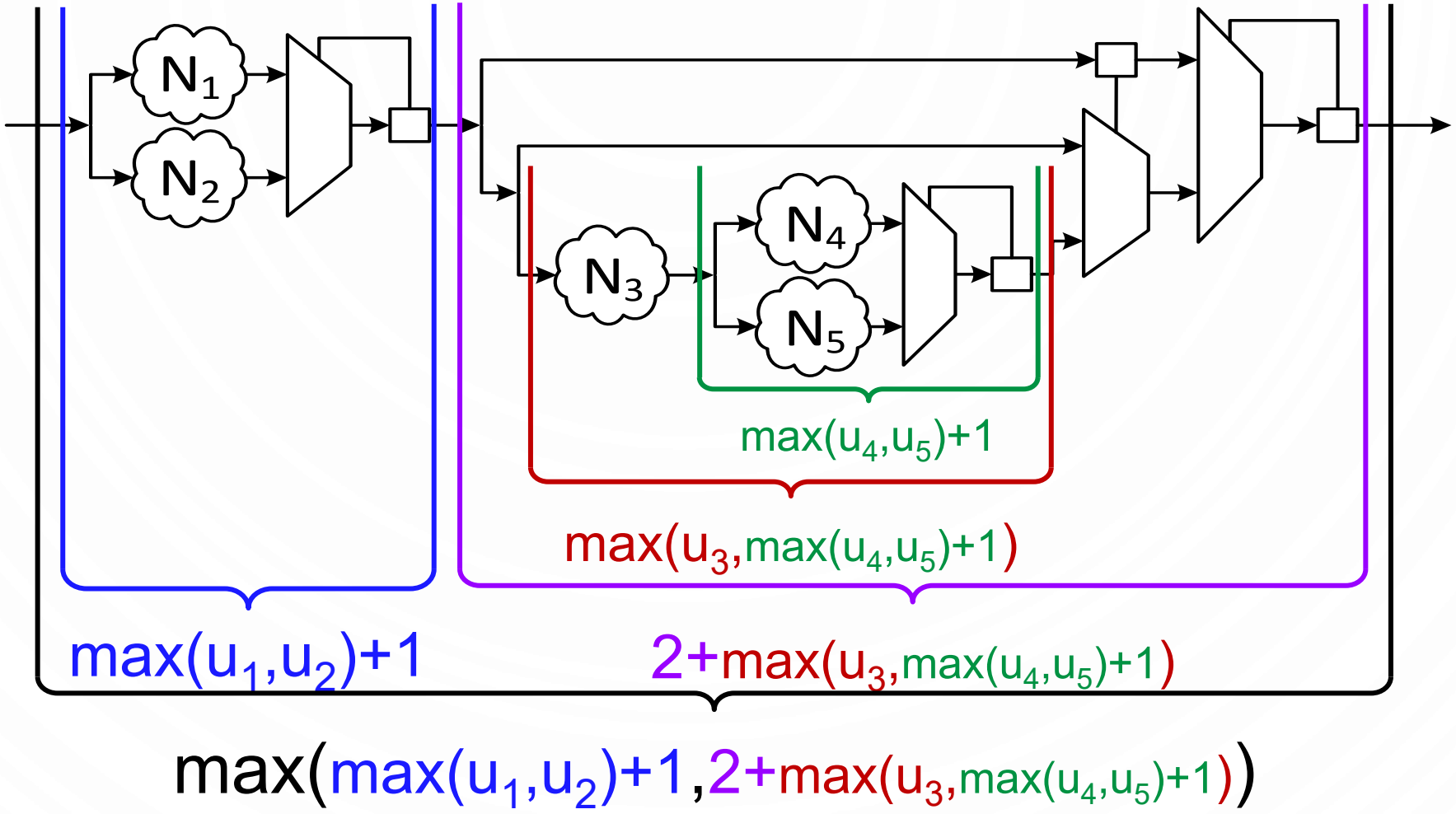
Lookup:
UCC(S) + 2



Rewriting:
max(UCC(S_1), UCC(S_2)) + 2

F. G. Zadegan, R. Krenz-Baath and E. Larsson, "Upper-bound computation for optimal retargeting in IEEE1687 networks," *2016 IEEE International Test Conference (ITC)*, 2016

Introduction to data structures in EDA – FSMs Decompose, Lookup, Rewriting



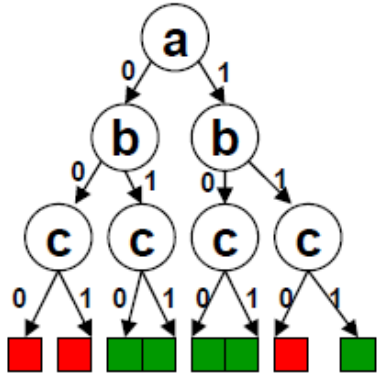
F. G. Zadegan, R. Krenz-Baath and E. Larsson, "Upper-bound computation for optimal retargeting in IEEE1687 networks," *2016 IEEE International Test Conference (ITC)*, 2016

SAT-CNF

- Given a Boolean formula (propositional logic formula), find a variable assignment such that the formula evaluates to 1, or prove that no such assignment exists.

$$F = (a + b)(a' + b' + c)$$

- For n variables, there are 2^n possible truth assignments to be checked.



- First established NP-Complete problem.

S. A. Cook, The complexity of theorem proving procedures, *Proceedings, Third Annual ACM Symp. on the Theory of Computing*,1971, 151-158

SAT-CNF

- Conjunctive Normal Form
  - F = (a + b)(a' + b' + c)

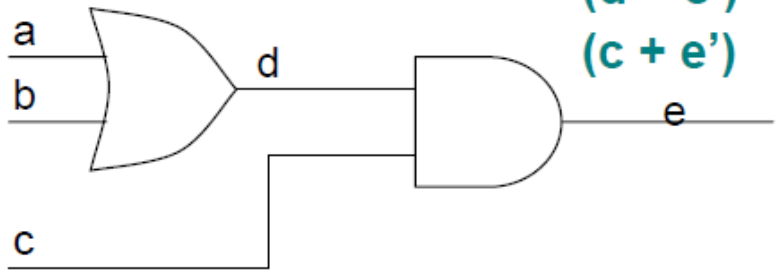
  - Simple representation (more efficient data structures)
- Logic circuit representation
  - Circuits have structural and direction information
- Circuit – CNF conversion is straightforward

d ≡ (a + b)

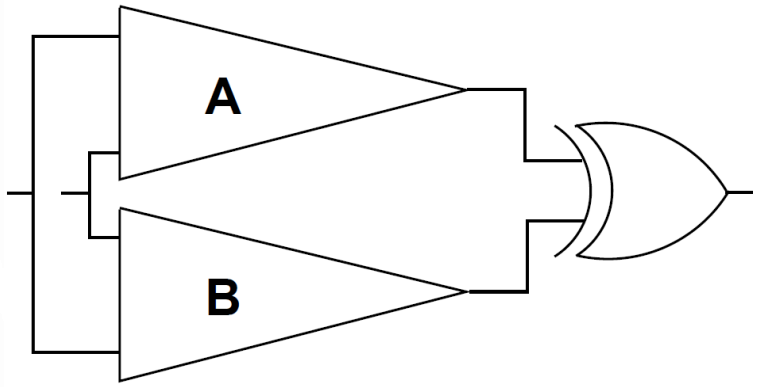
(a + b + d')
(a' + d)
(b' + d)

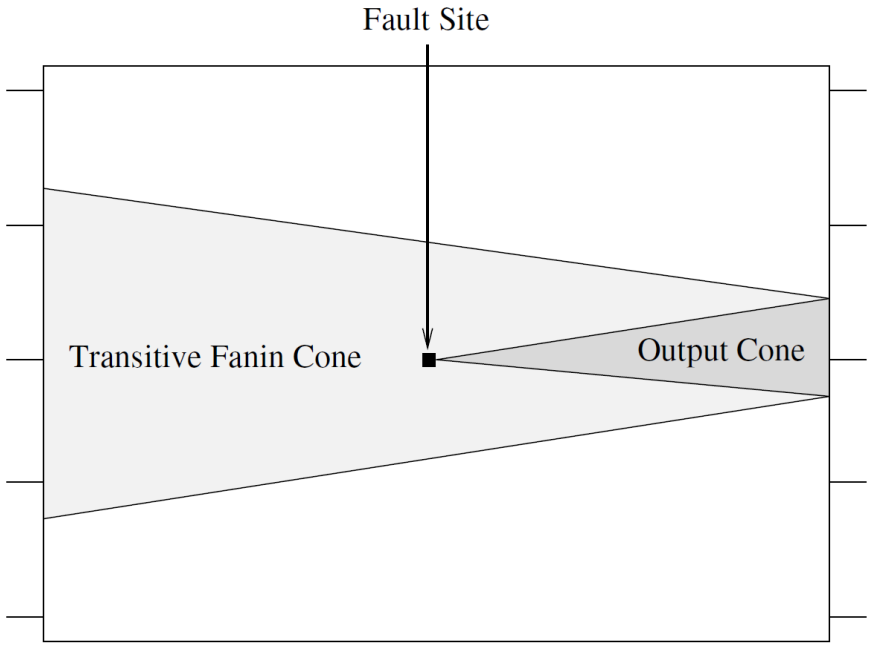
e ≡ (c · d)

(c' + d' + e)
(d + e')
(c + e')

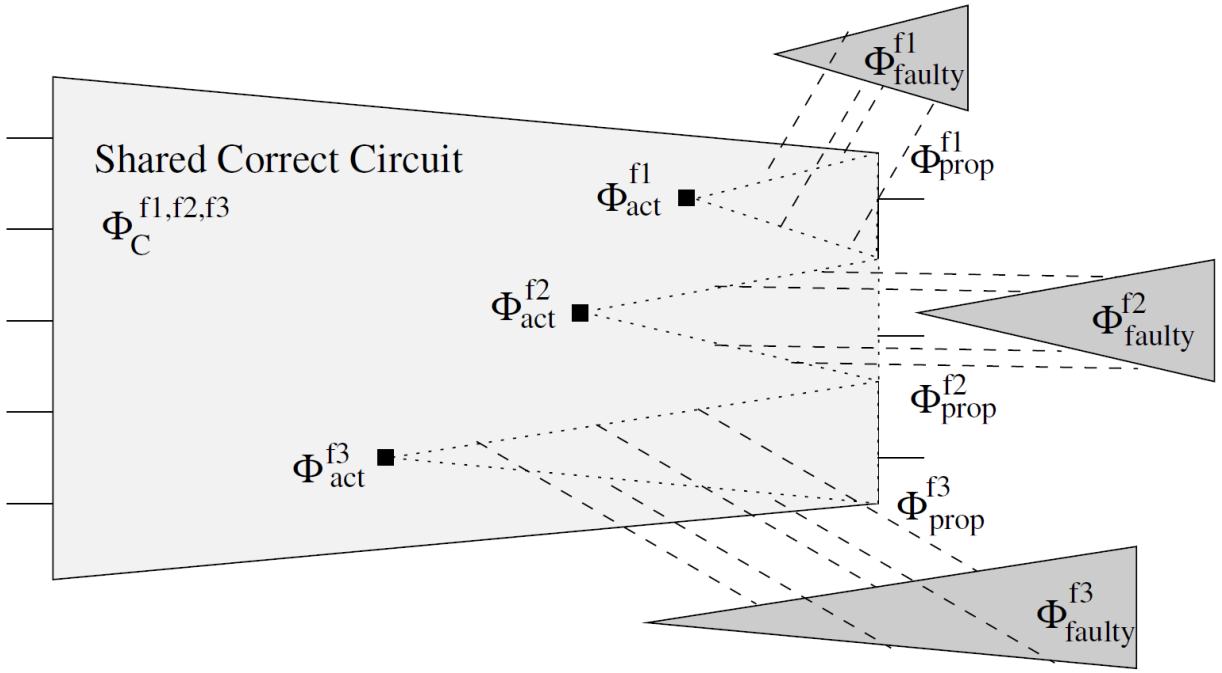


SAT-ATPG - Overview





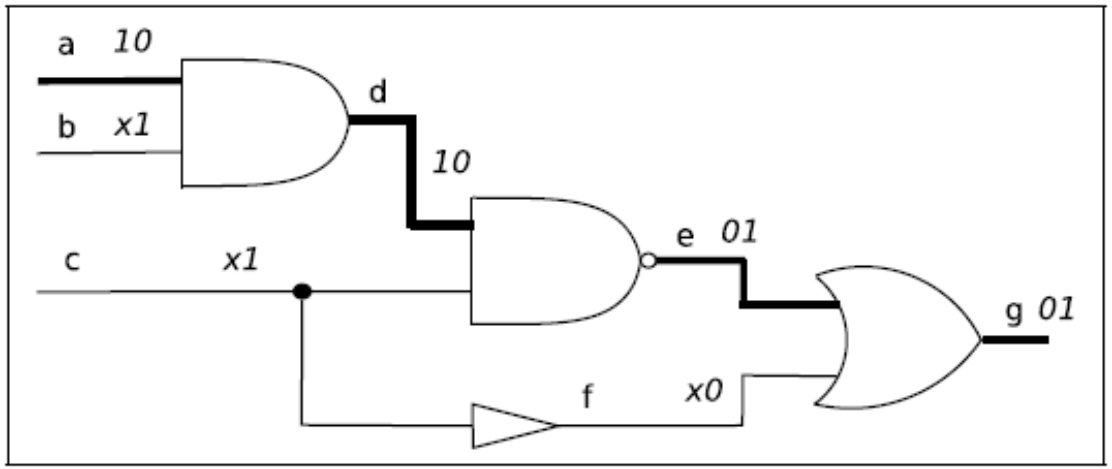
SAT-ATPG - Overview



SAT-ATPG

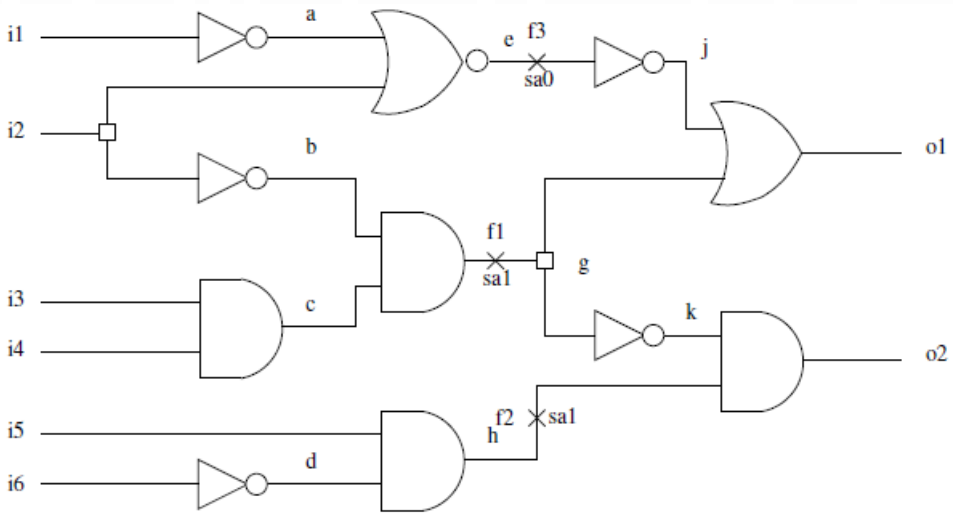
2-valued logic versus multi-valued logic

- #CNF-clauses 2-valued logic:
  - AND2: $(\overline{A} \vee \overline{B} \vee C) \wedge (A \vee \overline{C}) \wedge (B \vee \overline{C})$
  - XOR2: $(\overline{A} \vee \overline{B} \vee \overline{C}) \wedge (A \vee B \vee \overline{C}) \wedge (A \vee \overline{B} \vee C) \wedge (\overline{A} \vee B \vee C)$



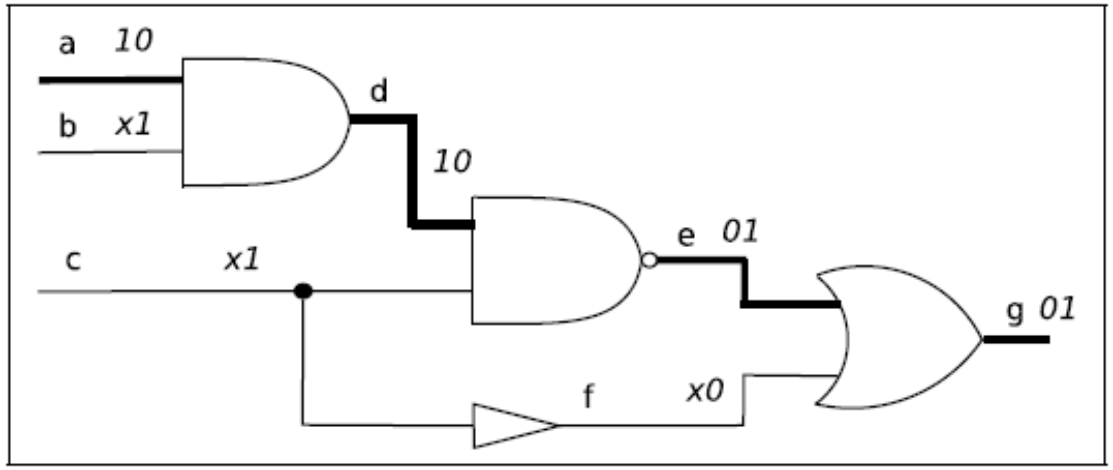
S. Eggersgluss et.al, "Combining Multi-Valued Logics in SAT-based ATPG for Path Delay Faults," *MEMOCODE'07.*

SAT-ATPG



	Good Circuit
$\Phi_a =$	$(a + i_1) \cdot (\overline{a} + \overline{i}_1)$
$\Phi_b =$	$(b + i_2) \cdot (\overline{b} + \overline{i}_2)$
$\Phi_c =$	$(c + \overline{i}_3 + \overline{i}_4) \cdot (\overline{c} + i_3) \cdot (\overline{c} + i_4)$
$\Phi_d =$	$(d + i_6) \cdot (\overline{d} + \overline{i}_6)$
$\Phi_e =$	$(e + a + i_2) \cdot (\overline{e} + \overline{a}) \cdot (\overline{e} + \overline{i}_2)$
$\Phi_g =$	$(g + \overline{b} + \overline{b}) \cdot (\overline{g} + b) \cdot (\overline{g} + c)$
$\Phi_h =$	$(h + \overline{i}_5 + \overline{d}) \cdot (\overline{h} + i_5) \cdot (\overline{h} + d)$
$\Phi_j =$	$(j + e) \cdot (\overline{j} + \overline{e})$
$\Phi_k =$	$(k + g) \cdot (\overline{k} + \overline{g})$
$\Phi_{o_1} =$	$(\overline{o}_1 + j + g) \cdot (o_1 + \overline{j}) \cdot (o_1 + \overline{g})$
$\Phi_{o_2} =$	$(o_2 + \overline{k} + \overline{h}) \cdot (\overline{o}_2 + k) \cdot (\overline{o}_2 + h)$
	Faulty Cone f_1
$\Phi_f^{f_1} =$	(g^{f_1})
$\Phi_{o_1}^{f_1} =$	$(\overline{o}_1^{f_1} + j + g^{f_1}) \cdot (o_1^{f_1} + \overline{j}) \cdot (o_1^{f_1} + \overline{g}^{f_1})$
$\Phi_k^{f_1} =$	$(k^{f_1} + g^{f_1}) \cdot (\overline{k}^{f_1} + \overline{g}^{f_1})$
$\Phi_{o_2}^{f_1} =$	$(o_2^{f_1} + \overline{k}^{f_1} + \overline{h}) \cdot (\overline{o}_2^{f_1} + k^{f_1}) \cdot (\overline{o}_2^{f_1} + h)$
	D-chain f_1
$\Phi_g^{Df_1} =$	$(\overline{g}^{Df_1} + g + g^{f_1}) \cdot (\overline{g}^{Df_1} + \overline{g} + \overline{g}^{f_1}) \cdot (\overline{g}^{Df_1} + k^{Df_1} + o_1^{Df_1})$
$\Phi_{o_1}^{Df_1} =$	$(\overline{o}_1^{Df_1} + o_1 + o_1^{f_1}) \cdot (\overline{o}_1^{Df_1} + \overline{o}_1 + \overline{o}_1^{f_1})$
$\Phi_k^{Df_1} =$	$(\overline{k}^{Df_1} + k + k^{f_1}) \cdot (\overline{k}^{Df_1} + \overline{k} + \overline{k}^{f_1}) \cdot (\overline{k}^{Df_1} + o_2^{Df_1})$
$\Phi_{o_2}^{Df_1} =$	$(\overline{o}_2^{Df_1} + o_2 + o_2^{f_1}) \cdot (\overline{o}_2^{Df_1} + \overline{o}_2 + \overline{o}_2^{f_1})$

SAT-based Pathdelay ATPG in Industrial Circuits



S. Eggersgluss et.al, "Combining Multi-Valued Logics in SAT-based ATPG for Path Delay Faults," *MEMOCODE'07.*

SAT-based Pathdelay ATPG in Industrial Circuits

$$L_4 = \{0, 1, U, Z\}$$

$$L_{6s} = \{0, \overline{0}, 01, 10, \overline{1}, 1\}$$

$$L_{8s} = \{0, \overline{0}, 01, 10, \overline{1}, 1, 0U, 1U\}$$

$$L_{11s} = \{0, \overline{0}, 01, 10, \overline{1}, 1, 0U, 1U, U0, U1, \overline{U}\}$$

$$L_{16} = \{\overline{0}, 01, 10, \overline{1}, 0Z, 1Z, Z0, Z1, 0U, 1U, U0, U1, ZU, UZ, \overline{Z}, \overline{U}\}$$

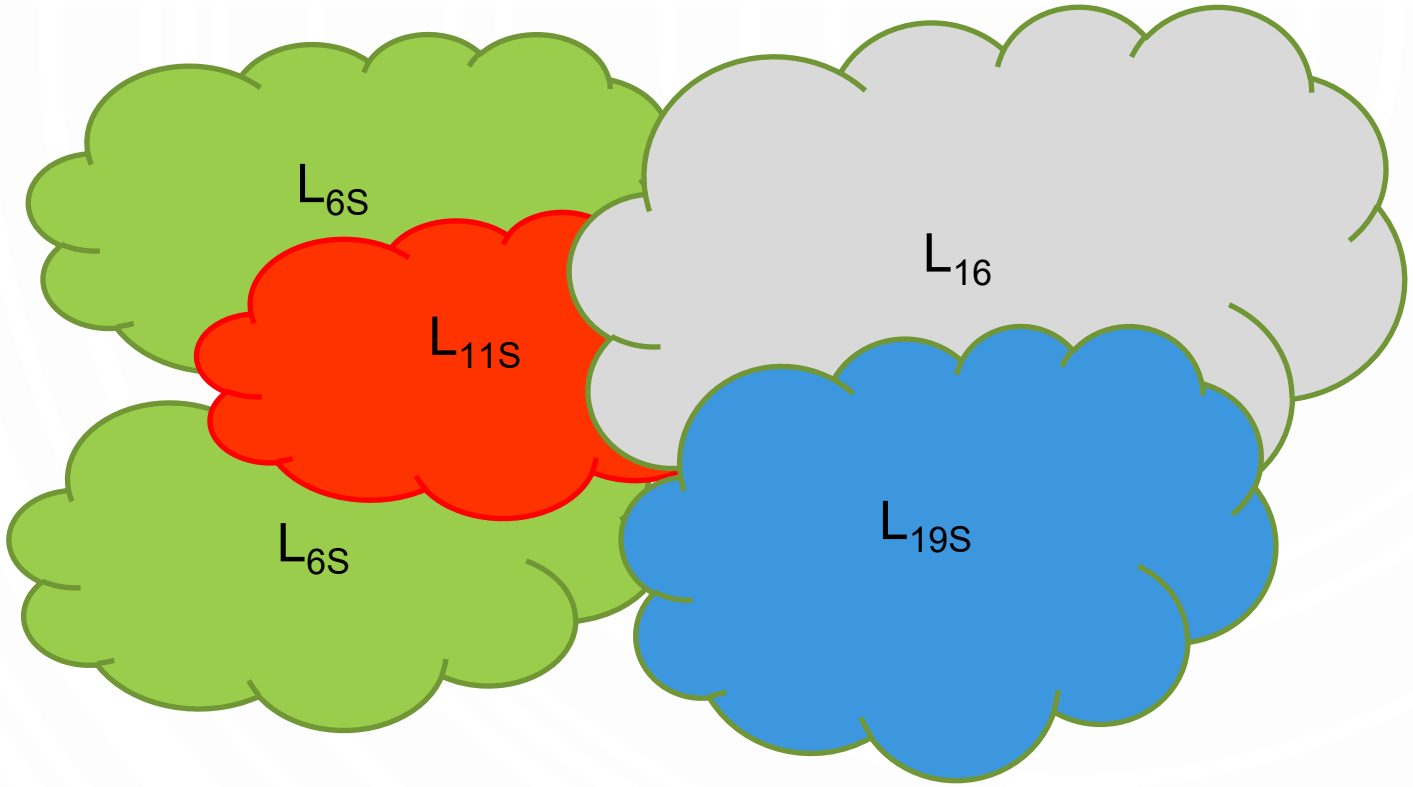
$$L_{19s} = \{0, \overline{0}, 01, 10, \overline{1}, 1, 0Z, 1Z, Z0, Z1, 0U, 1U, U0, U1, ZU, UZ, \overline{Z}, Z, \overline{U}\}$$

L_4: single-cycle time-frame industrial circuit
L_{6s}: two-cycle time-frame; no U-value and Z-value; static
L_{8s}: two-cycle time-frame; no Z-value, robust-pattern; static
L_{11s}: two-cycle time-frame; no Z-value, non-robust pattern; static
L_{16}: two-cycle time-frame; robust pattern
L_{19s}: two-cycle time-frame; non-robust pattern; static

		AND		bus-driver	
logic	# Var	# cls	# lits	# cls	# lits
L_{19s}	5	-	-	321	1794
L_{11s}	4	148	690	-	-
L_{8s}	3	38	152	-	-
L_{6s}	3	39	151	-	-
L_4	2	-	-	12	35
L_3	2	15	38	-	-
L_2	1	3	7	-	-

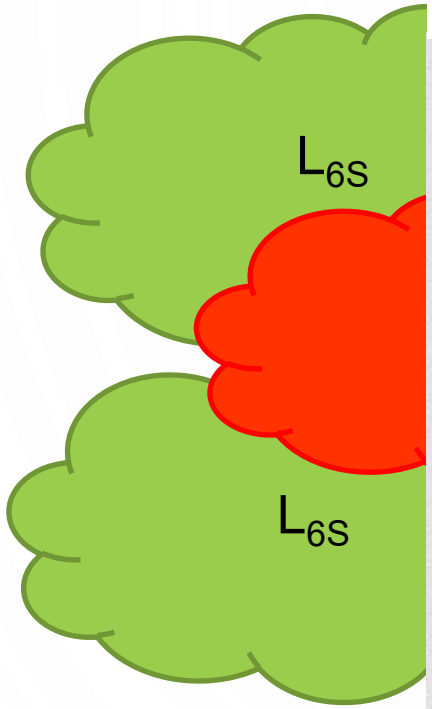
S. Eggersgluss et.al, "Combining Multi-Valued Logics in SAT-based ATPG for Path Delay Faults," *MEMOCODE'07.*

SAT-based Pathdelay ATPG in Industrial Circuits



S. Eggersgluss et.al, "Combining Multi-Valued Logics in SAT-based ATPG for Path Delay Faults," *MEMOCODE'07.*

SAT-based Pathdelay ATPG in Industrial Circuits



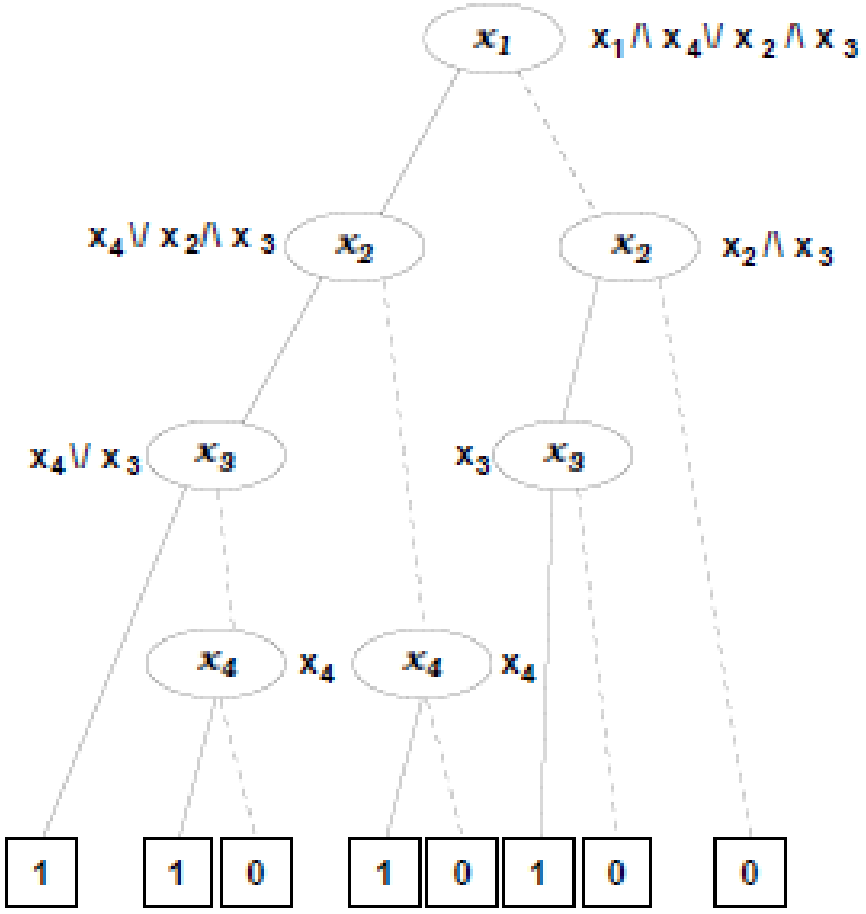
Lazy/Clever Preprocessing

```
132  bool Solver::addClause_(vec<Lit>& ps)
133  {
134      // Check if clause is satisfied and remove false/duplicate literals:
135      sort(ps);
136      Lit p; int i, j;
137      for (i = j = 0, p = lit_Undef; i < ps.size(); i++)
138          if (value(ps[i]) == l_True || ps[i] == ~p)
139              return true;
140          else if (value(ps[i]) != l_False && ps[i] != p)
141              ps[j++] = p = ps[i];
142      ps.shrink(i - j);
143
144      if (ps.size() == 0)
145          return ok = false;
146      else if (ps.size() == 1){
147          uncheckedEnqueue(ps[0]);
148          return ok = (propagate() == CRef_Undef);
149      }else{
150          CRef cr = ca.alloc(ps, false);
151          clauses.push(cr);
152          attachClause(cr);
153      }
154      return true;
155  }
```

S. Eggersgluss et.al, "Combining Multi-Valued Logics in SAT-based ATPG for Path Delay Faults," *MEMOCODE'07.*

Ordered Binary Decision Diagrams (OBDDs)

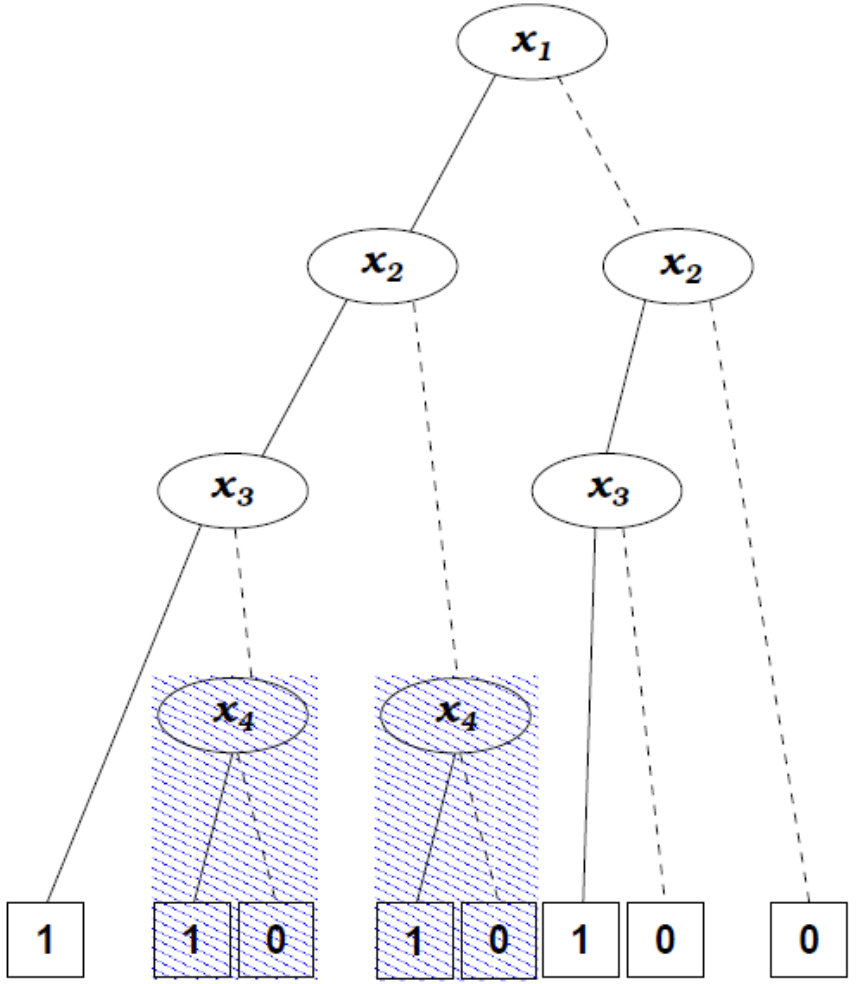
An OBDDs is an efficient graph-based representation of Boolean functions.

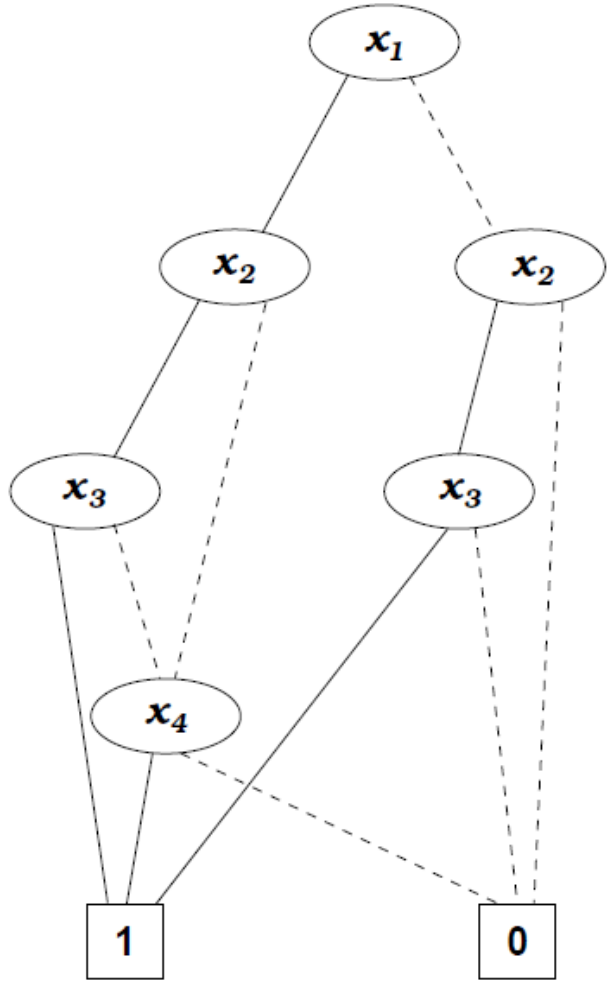


- **Terminal Nodes (Leaves):** There are terminal nodes, labeled **0 (FALSE)** and **1 (TRUE)**.
- **Non-Terminal Nodes (Decision Nodes):** Each non-terminal node $v \in V$ is labeled by a Boolean variable $var(v) \in \{x_1, x_2, \ldots, x_n\}$. Each non-terminal node has exactly two outgoing edges:
  - A **"low" edge** leading to a child node low(v), which corresponds to the case where var(v) is assigned the value **0**.
  - A **"high" edge** leading to a child node high(v), which corresponds to the case where var(v) is assigned the value **1**.
- For every path from root node to a terminal node, there exists a **global variable order** for any appearing node
- OBDDs are **not canonical**

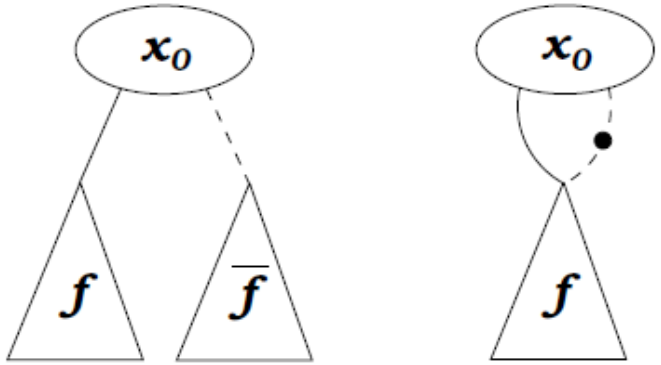
Reduced Ordered Binary Decision Diagrams (ROBDDs)

A ROBDD is an OBDD, whose isomorphic subtrees are merged.
ROBDDs are canonical !!!

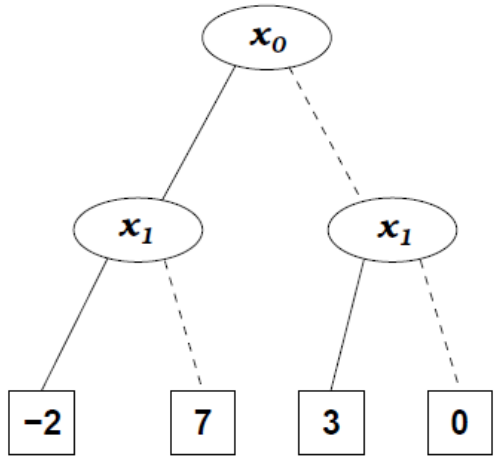




Reduced Ordered Binary Decision Diagrams (ROBDDs)



BDD using complemented edges



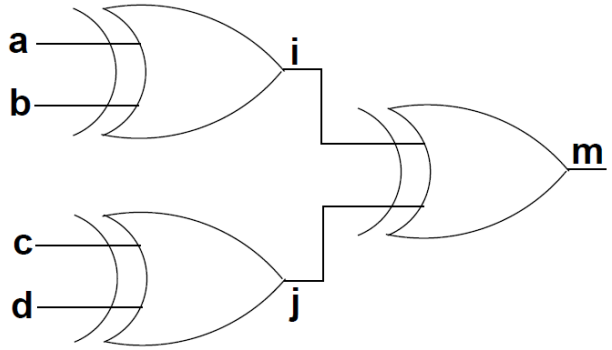
Multi-terminal BDD (MTBDDs)

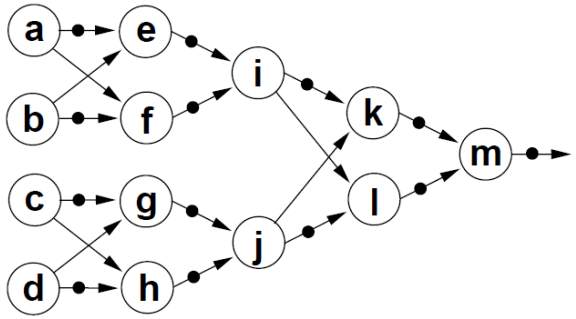
ROBDDs – Variable Recycling

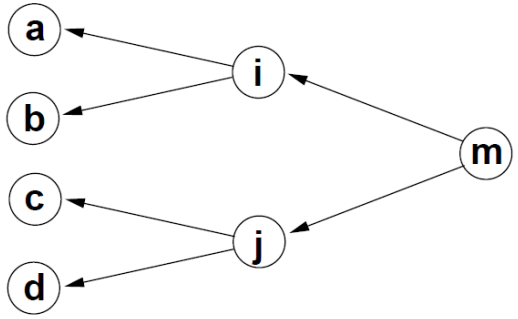
Observation:

- Many EDA-applications apply extensive decomposition strategies
- Majority runtime while applying ROBDDs spent on maintaining BDD-internal data structures (unique table ensuring canonicity)
- Reusing BDD-resources in different decomposed sections of the circuit

Variable-recycling reduces significantly size and maintenance effort of BDD-internal data structures.







ROBDDs – Variable Recycling

Observation:

- Many EDA-applications apply extensive decomposition strategies
- Majority runtime while applying ROBDDs spent on maintaining BDD-internal data structures (unique table ensuring canonicity)
- Reusing BDD-resources in different decomposed sections of the circuit

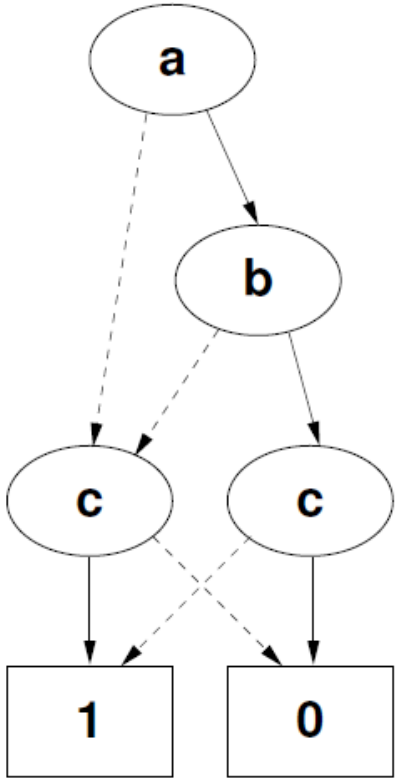
Variable-recycling reduces significantly size and maintenance effort of BDD-internal data structures.

number inputs	number vertices	without variable recycling			with variable recycling		
		MTBDD nodes	t,msec	deviation	MTBDD nodes	t,msec	deviation
64	99	1142	2.51	1.77 %	679	1.20	0.73 %
128	195	2295	5.00	0.85 %	1283	2.32	0.87 %
256	387	4590	10.42	0.46 %	2453	4.62	1.85 %
512	771	9166	22.28	0.52 %	4782	9.39	0.99 %
1024	1539	18271	51.70	20.72 %	9328	18.95	0.80 %
2048	3075	36150	156.63	0.36 %	18038	39.52	11.87 %
4096	6147	70991	619.51	0.92 %	34491	80.62	1.38 %
8192	12291	137265	2413.98	0.40 %	63904	171.26	1.31 %
16384	24579	259079	9445.99	0.26 %	111844	348.50	0.61 %
32768	49155	479227	39780.53	0.23 %	183882	743.35	2.69 %
65534	98307	-	-	-	291330	1521.03	0.73 %

Improved Circuit-to-CNF Transformation for SAT-based ATPG using Variable Recycling

Observation:

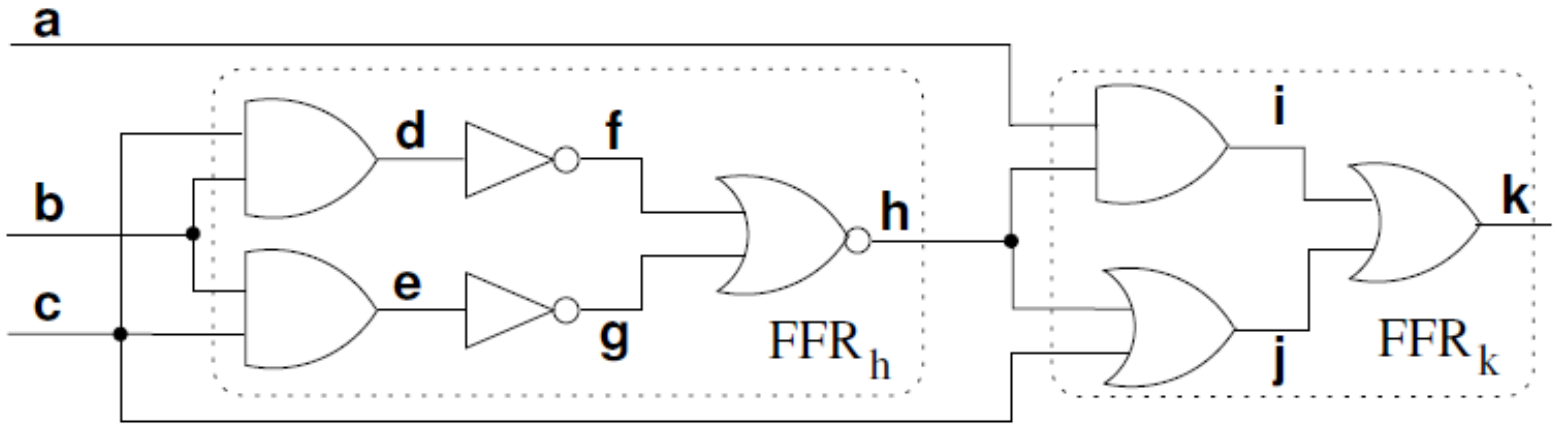
- CNF-generation in SAT-ATPG highly expensive
- Lookup-scheme for optimized and pre-generated CNF-clauses
- Fanout-free-regions (FFRs) are suitable for circuit partitioning
- Derive CNF-clauses directly from BDD-representation



$(\bar{a} + \bar{b} + \bar{c})$
$(\bar{a} + b + c)$
$(a + c)$

D. Tille, S. Eggersglüß, R. Krenz-Bååth, J. Schloeffel and R. Drechsler, "Improving CNF representations in SAT-based ATPG for industrial circuits using BDDs," ETS'10.

Improved Circuit-to-CNF Transformation for SAT-based ATPG using Variable Recycling



before

$$
\begin{array}{lll}
(d+\overline{b}+\overline{c}) & \cdot (\overline{d}+b) & \cdot (\overline{d}+c)\cdot \\
(e+\overline{b}+\overline{c}) & \cdot (\overline{e}+b) & \cdot (\overline{e}+c)\cdot \\
(h+f+g) & \cdot (\overline{h}+\overline{f}) & \cdot (\overline{h}+\overline{g})\cdot \\
(f+d) & \cdot (\overline{f}+\overline{d}\,)\cdot & \\
(g+e) & \cdot (\overline{g}+\overline{e}\,) &
\end{array}
$$

after

$$(h+\overline{b}+\overline{c})\cdot(\overline{h}+b)\cdot(\overline{h}+c)$$

D. Tille, S. Eggersglüß, R. Krenz-Bååth, J. Schloeffel and R. Drechsler, "Improving CNF representations in SAT-based ATPG for industrial circuits using BDDs," ETS'10.

Questions ?