Automatic Test Pattern Generation

Rolf Drechsler, Görschwin Fey
University of Bremen

drechsle@informatik.uni-bremen.de

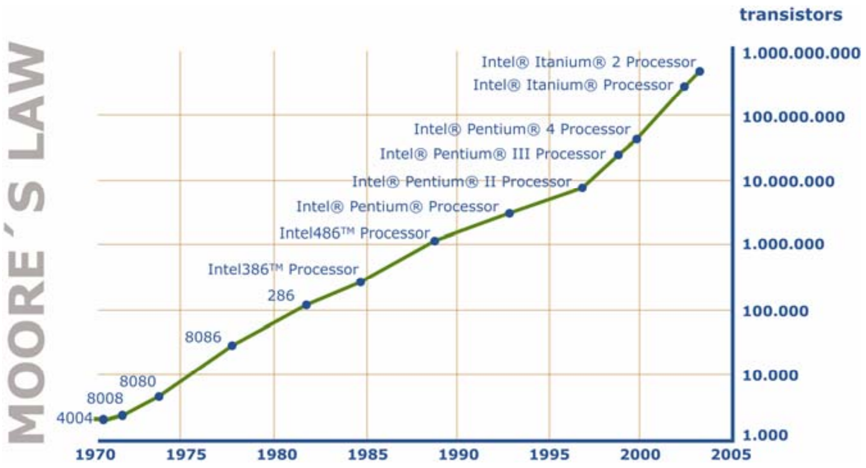
Universität Bremen - Computer Architecture

Outline

- Introduction/Motivation
- Preliminaries
  – Circuit, Fault Model, Test Pattern Generation
- Proof techniques
  – Boolean satisfiability, BDD, SAT, Circuit to SAT Conversion
- SAT-based ATPG
  – Problem description
  – Multi-valued Encoding
  – Variable Selection
- Experimental Results
- Conclusions

Universität Bremen - Computer Architecture

Motivation



Source: Intel

Universität Bremen - Computer Architecture

Motivation

- Increasing size of circuits
- Post-production test is a crucial step:
  - Have there been problems during production?
  - Does the circuit contain faults?
- Test patterns are applied

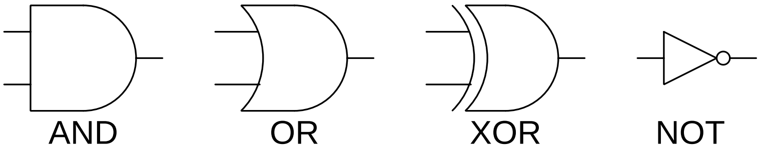
Motivation

- Test pattern generation happens at the Boolean level
- Classical ATPG algorithms reach their limits

➢ There is a need for more efficient ATPG tools!

Circuits

- Basic gates
  – AND, OR, EXOR, NOT



Fault Model

- Model “realistic” fault
  - Physical faults or defects at the Boolean level
- Simplified assumption
- Based on netlist

- Static or dynamic
  - Here: static only

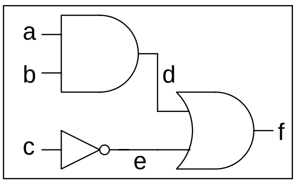
Universität Bremen - Computer Architecture

Stuck-at Fault Model

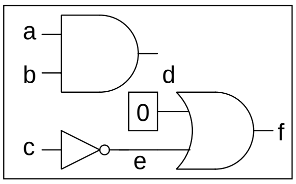
- Single line is assumed to have a fixed value (0 or 1)

- Example: stuck-at 0 fault at line d

correct

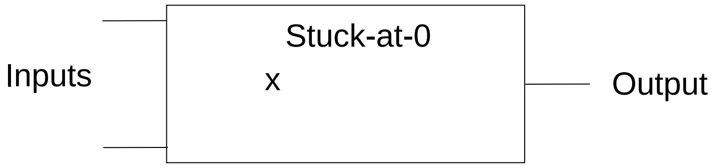


faulty



Test Pattern Generation

- Physical defects are modeled on the Boolean level



- Automatic Test Pattern Generation (ATPG)

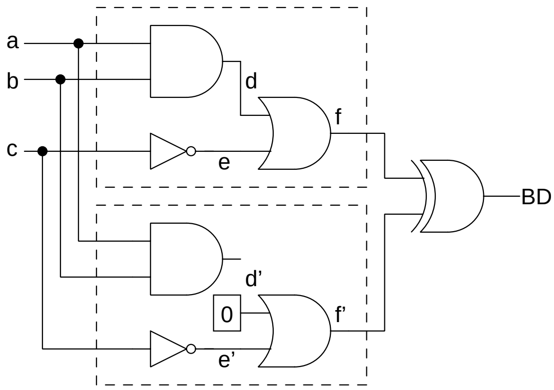
  Given: Circuit C and Fault-Model F

  Objective: Calculate test patterns for faults in C with respect to F

Universität Bremen - Computer Architecture

Boolean Difference

- BD of faulty and fault free circuit



Fault Classification

- If there is a test, the fault is *testable.*
- If there does not exist a test, the fault is *redundant*.

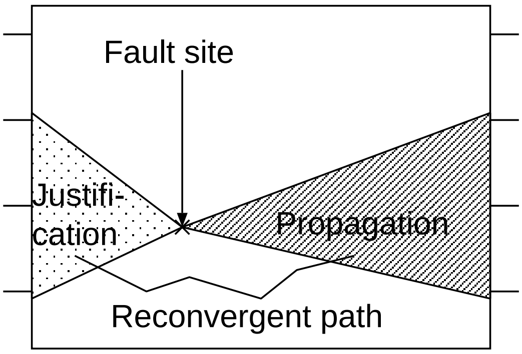
- Decision is NP complete.

ATPG: D-Algorithm

- An error is observed due to differing values at a line in the circuit with or without failure. Such a divergence is denoted by values D or D´ to mark differences 1/0 or 0/1, respectively.
- Instead of Boolean values, the set {0,1,D,D´} is used to evaluate gates and carry out implications.
- A gate that is not on a path between the error and any output does never have a D-value.
- A necessary condition for testability is the existence of a path from the error to an output, where all intermediate gates either have a D-value or are not assigned yet. Such a path is called a potential D-chain.
- A gate is on a D-chain, if it is on a path from the error location to an output and all intermediate gates have a D-value.

General Structure

- Justification and Propagation



Improvements

- PODEM: only branch on inputs
- FAN: branching on fanout stems
- SOCRATES: learning
- HANIBAL: recursive learning

- Alternative: SAT-based
  - Formulation based on formal techniques
  - Proof techniques: BDD and SAT

Representation

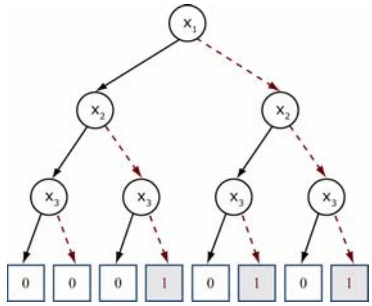
x_1	x_2	x_3	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

- Truth table
- SoP (DNF) and PoS (CNF)
- Examples
  - Sum-of-products
    $F = x_1'x_2x_3 + x_1x_2'x_3 + x_1x_2x_3$
  - Product-of-sums
    $F = (x_1+x_2+x_3)\,(x_1+x_2+x_3')$
    $(x_1+x_2'+x_3)\,(x_1'+x_2+x_3)\,(x_1'+x_2'+x_3)$
- Decision tree

Truth Table and Decision Tree

x_1	x_2	x_3	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

x_1	0	0	0	0	1	1	1	1
x_2	0	0	1	1	0	0	1	1
x_3	0	1	0	1	0	1	0	1
F	0	0	0	1	0	1	0	1

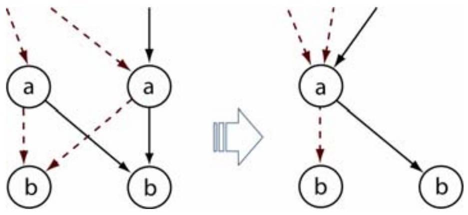


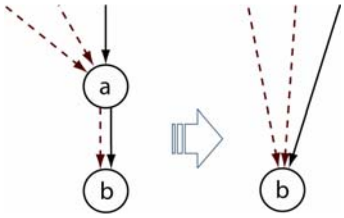
Universität Bremen - Computer Architecture

Reduction of Decision Tree

Rule 1: **Isomorphism Rule**
Nodes must be unique
(I-reduction)

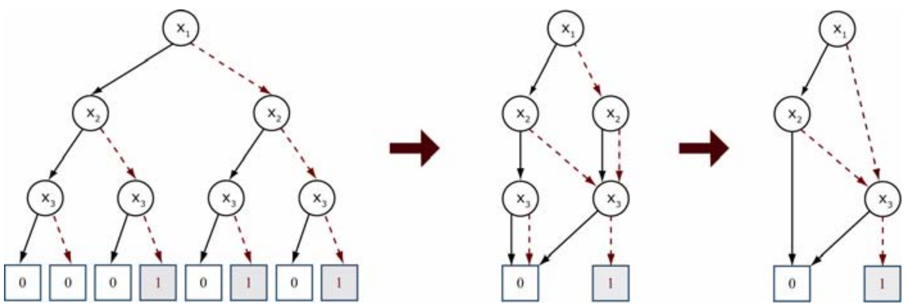
Rule 2: **Elimination Rule**
Redundant tests should not
be present (S-reduction)





Universität Bremen - Computer Architecture

Example of Tree Reduction



Decision Tree ⇒ reduction BDD

Shannon Expansion

- A Boolean function can be expanded by Shannon

$$F(x,y,z) = x' F_{x'} + x F_x$$

where $F_{x'}$ and F_x are positive (negative) cofactors

$$F_{x'} = F(0, y, z),\ F_x = F(1, y, z)$$

Synthesis Operations: ITE

- If-Then-Else-Operator:

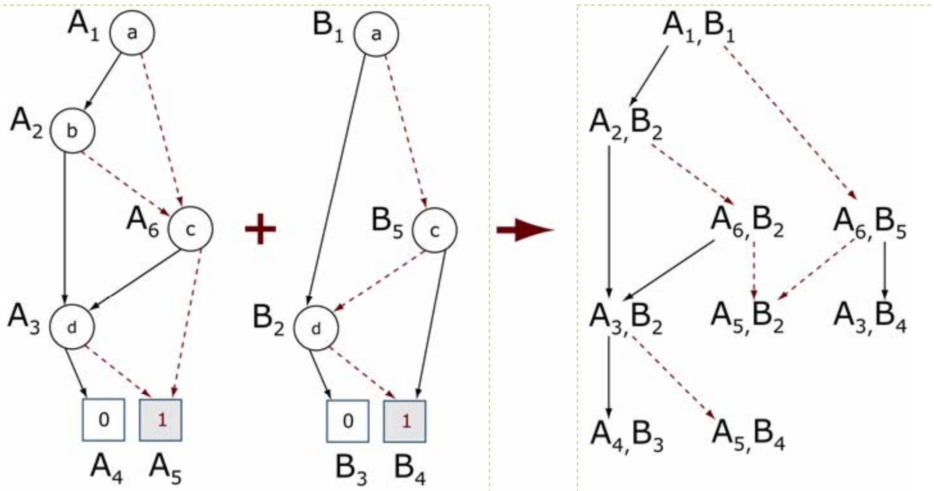
$$ITE(F, G, H) = F\,G + F'\,H$$

- Boolean operations over ITE arguments can be expressed as ITE of F, G, and constants
- Example: AND(F, G) = ITE(F, G, 0)
- Computation of Boolean operations is based on the Shannon expansion:

$$ITE(F,G,H) = ITE(x,\ ITE(F_{x'},G_{x'},H_{x'}),\ \ ITE(F_x,G_x,G_x))$$

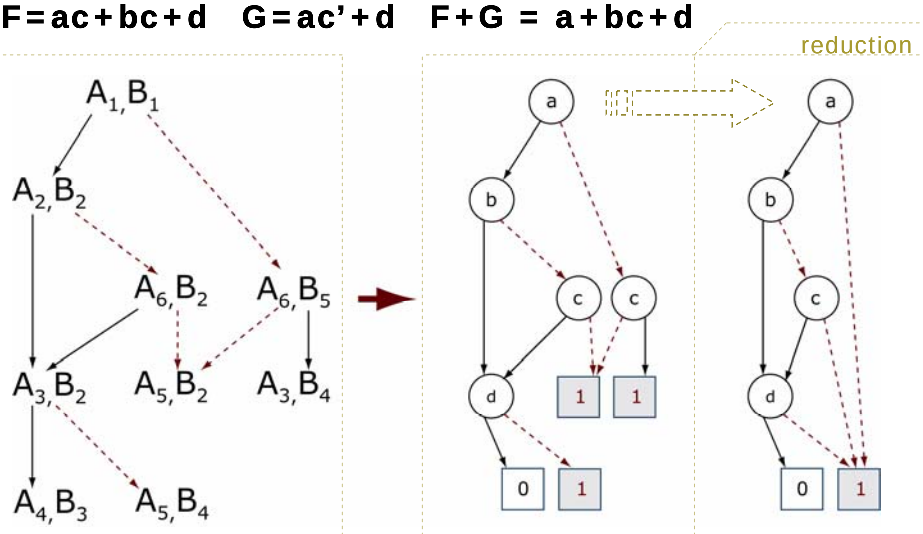
Example:

F=ac+bc+d   G=ac'+d   F+G = ?



Example:

F=ac+bc+d   G=ac'+d   F+G = a+bc+d



Universität Bremen - Computer Architecture

Properties

- Efficient implementation
- Compact representation for many Boolean functions
- Polynomial manipulation algorithms
- Sensitive to variable ordering
  - NP-complete problem
  - Dynamic variable ordering

23

Universität Bremen - Computer Architecture

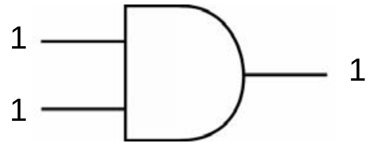
Function Representation

- BDD-based representation of
  - functions (with don’t cares)
  - relations
  - minterms, cubes
  - sets (of sets)
  - state machines
  - …
- Common features of all successful BDD-based representations

24

Simulation

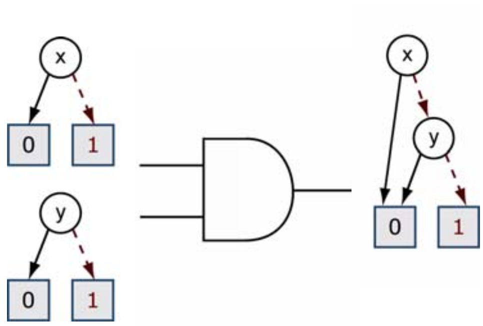
- Application of values
- Fast computation
  - linear time
- New evaluation for each input pattern
- Complete simulation only feasible for small circuits
  - exponential in the number of inputs



Universität Bremen - Computer Architecture

Symbolic Simulation

- Application of variables
- One computation for all input patterns in parallel
- Construction of diagrams for each gate
  - synthesis operations
- Size of diagrams

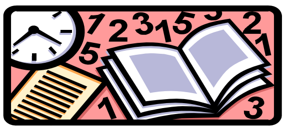


SAT

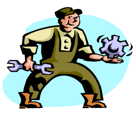
- Often all patterns are not needed
- A single test-vector is sufficient
- Construction of satisfying assignment

- SAT-problem: For a given Boolean function f find an assignment a, such that f(a)=1 or prove that such an assignment does not exist.

SAT

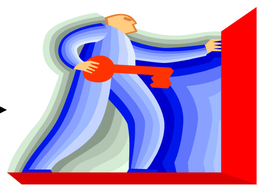


"real problem"



"real solution"

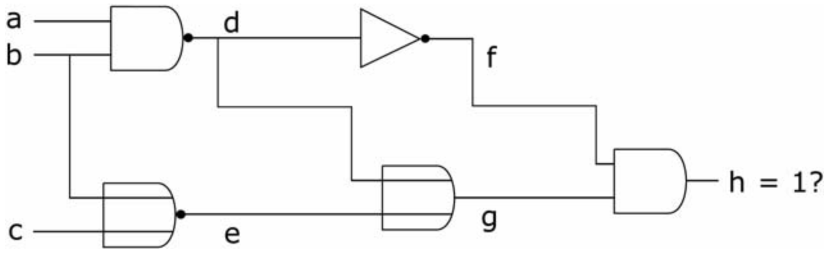
$$(a+b+\overline{c})\cdot(\overline{a}+c)\cdot(\overline{b}+c)\cdot(c+\overline{d})\cdot(\overline{c}+d)\cdot(\overline{d}+\overline{e}+f)\cdot(d+\overline{f})\cdot(e+\overline{f})\cdot f$$



$a=0, b=0, c=0, d=1, e=1, f=1$

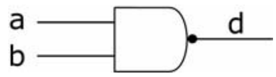
SAT instance → SAT solver → SAT solution

SAT for Circuits



$\varphi = h\ [d = \neg(ab)]\ [e = \neg(b + c)]\ [f = \neg d]\ [g = d + e]\ [h = fg]$

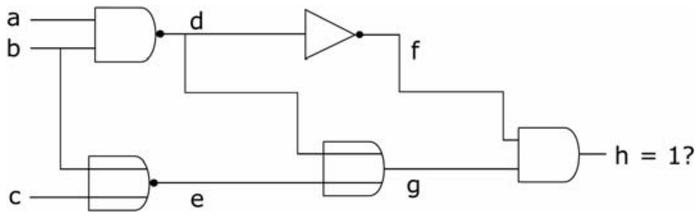
CNF of a Gate



$$\varphi_d = [d = \neg(a\ b)]$$

$$= \neg[d \oplus \neg(a\ b)]$$
$$= \neg[\neg(a\ b)\neg d + a\ b\ d]$$
$$= \neg[\neg a\ \neg d + \neg b\ \neg d + a\ b\ d]$$
$$= (a + d)(b + d)(\neg a + \neg b + \neg d)$$

CNF for Circuit



$\varphi = h\ [d = (ab)]\ [e = \neg(b + c)]\ [f = \neg d]\ [g = d + e]\ [h = fg]$

$= h$

$(a + d)(b + d)(\neg a + \neg b + \neg d)$

$(\neg b + \neg e)(\neg c + \neg e)(b + c + e)$

$(\neg d + \neg f)(d + f)$

$(\neg d + g)(\neg e + g)(d + e + \neg g)$

$(f + \neg h)(g + \neg h)(\neg f + \neg g + h)$

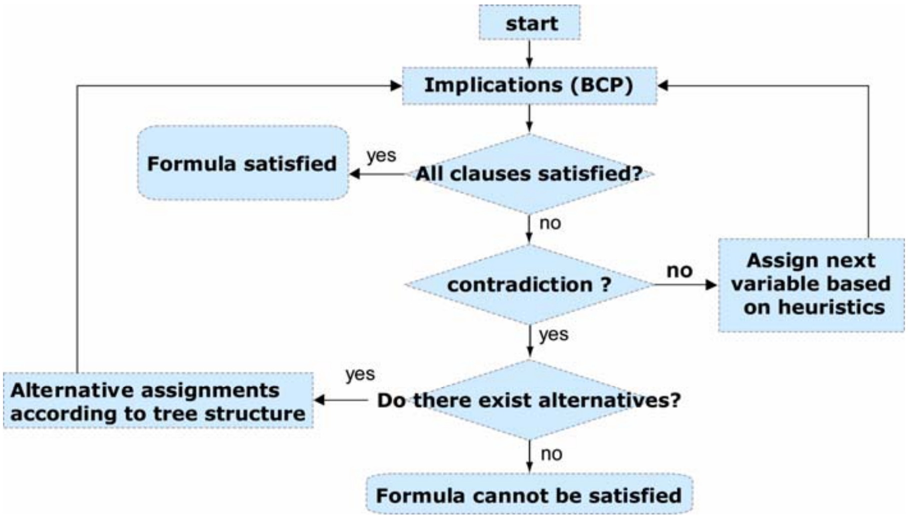
- CNF for circuit and assignment h=1

SAT Solving

- Most Algorithms are based on DLL procedure

- Overall flow
  - Assign variables in the CNF
  - If a contradiction occurs backtrack

Basic Procedure

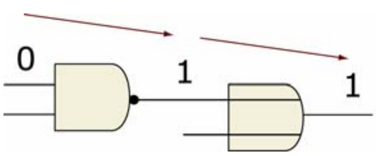


Implications

- Unit clause: Only one unspecified literal

$$(\neg a + b + \neg c) \quad \underset{1}{a} = 1,\ \underset{0}{b} = 0 \quad \Rightarrow c = 0$$

- Boolean constraint propagation (BCP) is based on iteration of unit clause rule
- BCP corresponds to implications on the net list
- Fast implementation, since CNF is very regular

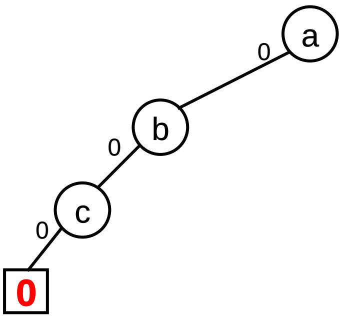


Reasons for SAT Efficiency

- Implications
- Analysis of backtracks
- Decision heuristics
- Conflict learning
  - Instance grows
- Non-chronological backtracking
- Data structure
  - CNF
  - Circuit

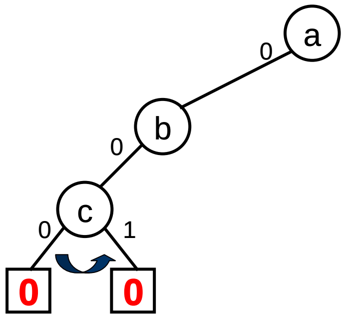
DLL – An Example

$(a+c+d) \cdot$
$(a+c+\overline{d}) \cdot$
$(a+\overline{c}+d) \cdot$
$(a+\overline{c}+\overline{d}) \cdot$
$(\overline{a}+b+c+d)$



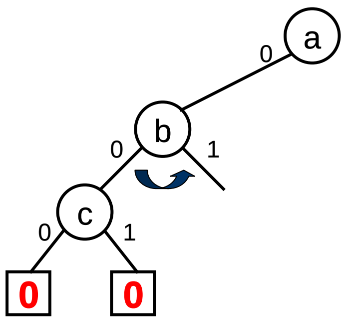
DLL – An Example

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



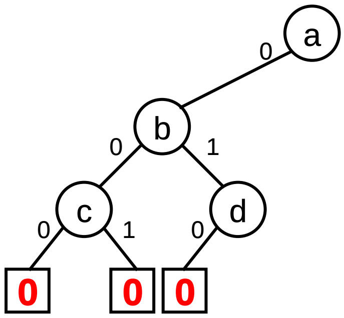
DLL – An Example

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



DLL – An Example

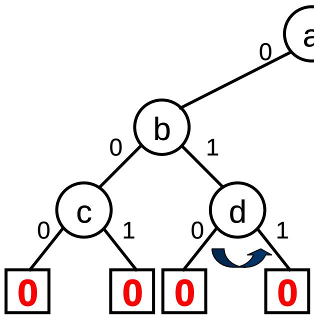
$(a+c+d)\cdot$
$(a+c+\overline{d})\cdot$
$(a+\overline{c}+d)\cdot$
$(a+\overline{c}+\overline{d})\cdot$
$(\overline{a}+b+c+d)$



Universität Bremen - Computer Architecture

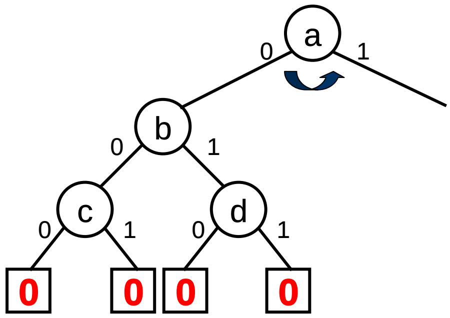
DLL – An Example

$(a+c+d)\cdot$
$(a+c+\overline{d})\cdot$
$(a+\overline{c}+d)\cdot$
$(a+\overline{c}+\overline{d})\cdot$
$(\overline{a}+b+c+d)$



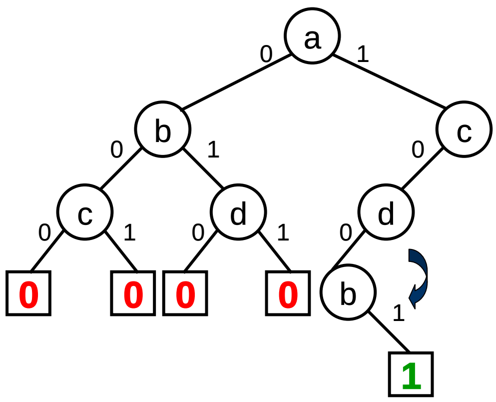
DLL – An Example

$(a+c+d) \cdot$
$(a+c+\overline{d}) \cdot$
$(a+\overline{c}+d) \cdot$
$(a+\overline{c}+\overline{d}) \cdot$
$(\overline{a}+b+c+d)$



DLL – An Example

$(a+c+d) \cdot$
$(a+c+\overline{d}) \cdot$
$(a+\overline{c}+d) \cdot$
$(a+\overline{c}+\overline{d}) \cdot$
$(\overline{a}+b+c+d)$



BDDs versus SAT

- BDDs consider all solutions
- SAT finds single solution
- Backtrack tree similar to BDD structure

- Advanced SAT techniques:
  - Variable selection strategies
  - Efficient implementations
    - Engineering
  - Implications
  - Conflict analysis

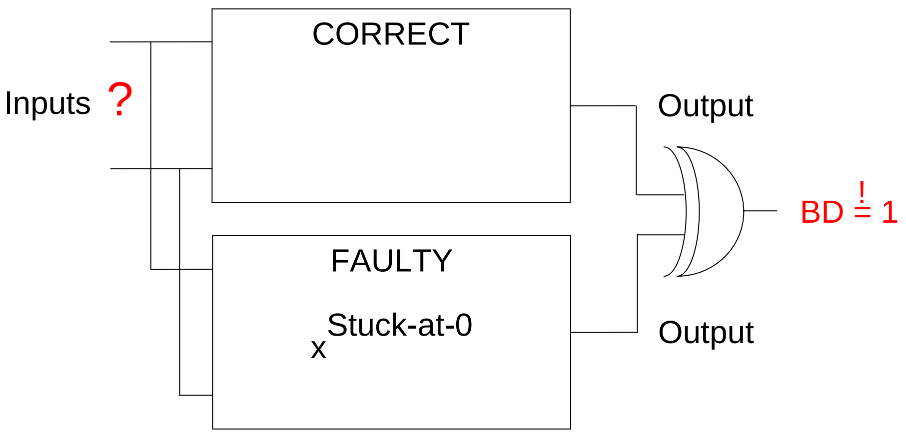
Motivation for SAT-based ATPG

- Substantial improvements in SAT solving

➢ Use
  - Advanced SAT techniques
  - In combination with structural information

  For
  - Large industrial circuits
  - In a multi-valued domain

Test Pattern Generation



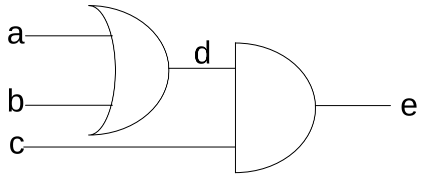
SAT-based ATPG

- **Input:** Circuit C, Fault F

1. Fault modeling:
   BD between fault free and faulty circuit
2. Translate into CNF
3. Use SAT solver to calculate solution

- **Output:** Classification of F, Testvector T

Circuit → CNF

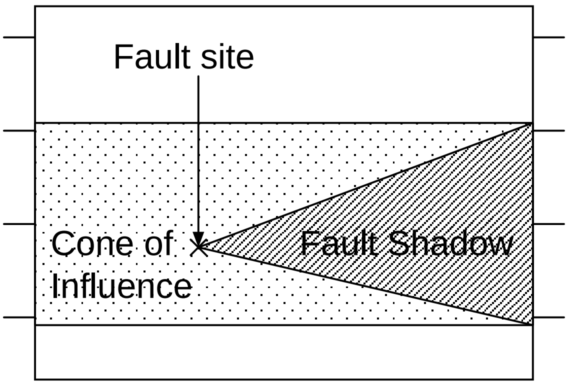


- AND-gate: (c'+d'+ e) · (c+e') · (d+e')
- OR-gate: (a+b+d') · (a'+d) · (b'+d)

- Linear size conversion

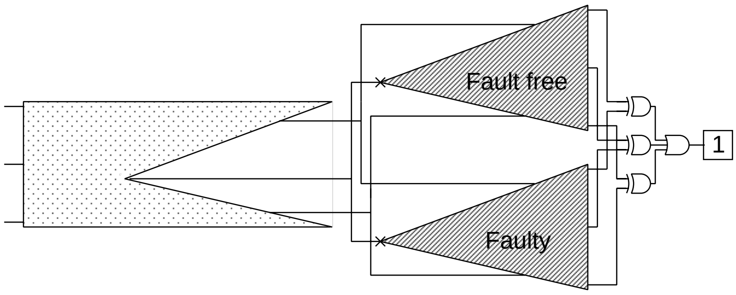
Use of Structural Information

- Influenced circuit parts

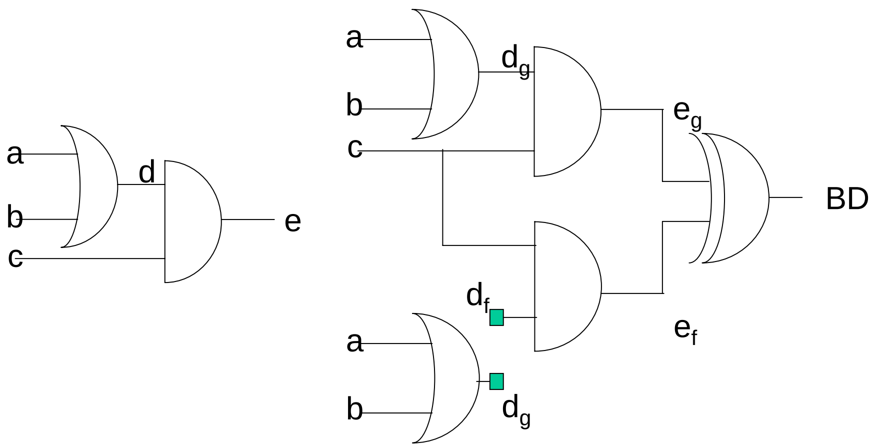


Create Instance

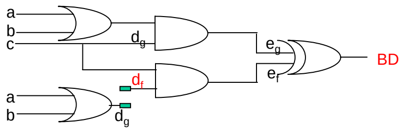
- Build circuit structure accordingly



Fault modeling



CNF

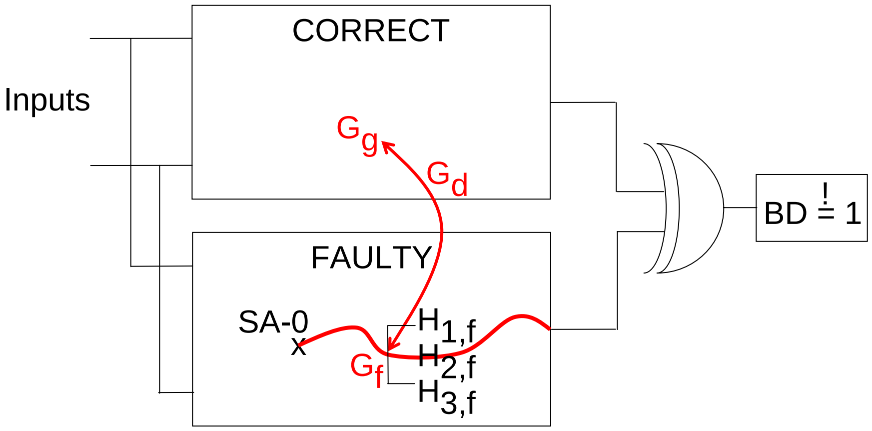


- $F = (\overline{c} + \overline{d}_g + e_g) \cdot (c + \overline{e}_g) \cdot (d_g + \overline{e}_g)$
  $\cdot (a + b + \overline{d}_g) \cdot (\overline{a} + d_g) \cdot (\overline{b} + d_g) \cdot (d_f)$
  $\cdot (\overline{c} + \overline{d}_f + e_f) \cdot (c + \overline{e}_f) \cdot (d_f + \overline{e}_f)$
  $\cdot (e_g + e_f + \overline{BD}) \cdot (\overline{e}_g + \overline{e}_f + \overline{BD})$
  $\cdot (\overline{e}_g + e_f + BD) \cdot (e_g + \overline{e}_f + BD) \cdot (BD)$
- F is the CNF for circuit with d s-a-1
- Inputs satisfy CNF $\rightarrow$ can detect fault
- CNF is linear in circuit size

Structural information: TEGUS

- Use approach as in **D-algorithm**
- Gate G on path between fault and output:
  - unfaulty circuit: $G_g = G(X_g)$
  - faulty circuit: $G_f = G(X_f)$
- G on a D-chain implies
  - difference: $G_d \rightarrow (G_f \neq G_g)$
  - at least one successor is on the D-chain: $G_d \rightarrow (H_{1,d} + \ldots + H_{k,d})$

Structural Information: TEGUS



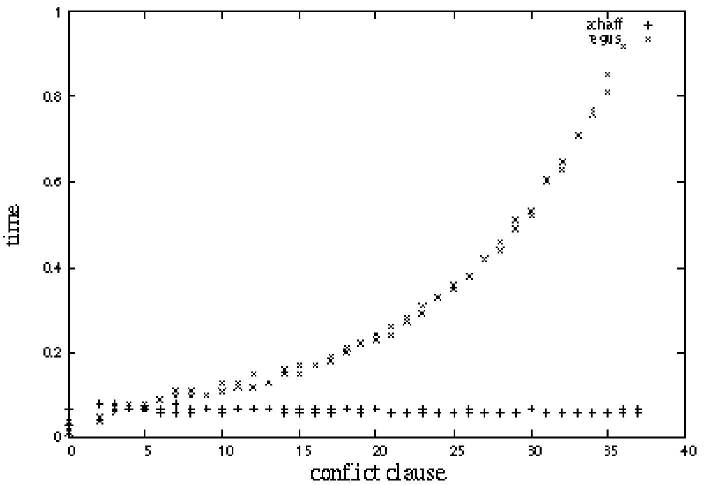
Features of PASSAT

- Memory Management
- Advanced SAT techniques
- Problem specific variable selection
- Multi-valued model

Advanced SAT Techniques

- Built-in techniques from Zchaff
  - Conflict based learning
  - Non-chronological backtracking
  - Event-driven evaluations
  - Clever decision heuristics

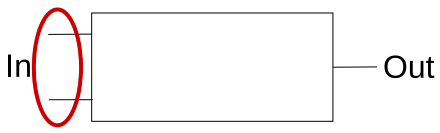
Advanced SAT Techniques:
SAT Run Times for Redundant Faults



Variable Selection

- Use problem specific strategies to chose next decision variable

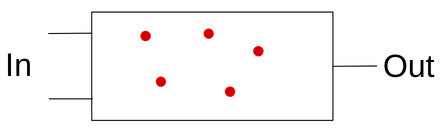
- Only inputs
- Only fanouts
- Zchaff´s default strategy
- Combined strategy
  - First: Only inputs with time-out
  - Then: Zchaff´s default



Variable Selection

- Use problem specific strategies to chose next decision variable

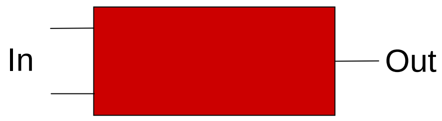
- Only inputs
- Only fanouts
- Zchaff´s default strategy
- Combined strategy
  - First: Only inputs with time-out
  - Then: Zchaff´s default



Universität Bremen - Computer Architecture

Variable Selection

- Use problem specific strategies to chose next decision variable



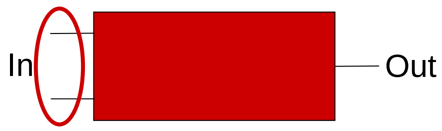
- Only inputs
- Only fanouts
- Zchaff´s default strategy
- Combined strategy
  - First: Only inputs with time-out
  - Then: Zchaff´s default

59

Universität Bremen - Computer Architecture

Variable Selection

- Use problem specific strategies to chose next decision variable



- Only inputs
- Only fanouts
- Zchaff´s default strategy
- Combined strategy
  - First: Only inputs with time-out
  - Then: Zchaff´s default

60

Variable Selection: p49k

Heuristic	Cnt	Red	Ab	Time(s)
All	0	1	255	3847
Input	187	67	2	1787
Fanout	0	0	256	2568
Input+All	187	68	1	2084

Multi-valued Model

- Application to industrial circuits
- Allow for ‚Z´ and ‚U´ values
- Encode circuit lines by two variables
- Optimize the encoding

Multi-valued Model

- Encoding

X	Encode		Interpretation
	c_x	c^*_x	
0	0	0	Signal X is 0
1	1	0	Signal X is 1
U	1	1	Signal X is unknown
Z	0	1	Signal X is at 'Z'

- Clauses for c = a • b

$$(\overline{c}_a+\overline{c}_b+c_c)\cdot(c_a^*+c_b^*+\overline{c}_c^*)\cdot(c_c+\overline{c}_c^*)\cdot$$
$$(c_a+c_a^*+\overline{c}_c)\cdot(c_b+c_b^*+\overline{c}_c)\cdot(\overline{c}_a^*+\overline{c}_b+c_c^*)\cdot$$
$$(\overline{c}_a+\overline{c}_b^*+c_c^*)\cdot(\overline{c}_a^*+\overline{c}_b^*+c_c^*)$$

Multi-valued Model: Encodings

s	x	$\overline{x}$
0	a	b
1	a	$\overline{b}$
U	$\overline{a}$	b
Z	$\overline{a}$	$\overline{b}$

s	x	$\overline{x}$
0	a	b
1	a	$\overline{b}$
U	$\overline{a}$	$\overline{b}$
Z	$\overline{a}$	b

s	x	$\overline{x}$
0	a	b
1	$\overline{a}$	$\overline{b}$
U	$\overline{a}$	b
Z	a	$\overline{b}$

„natural“

Multi-valued Model: Encodings

circ.	enc.	clauses	cls. %	CNF	CNF %	solve	solve %
p44k	A B	174,083 220,493	127	43 52	121	15 79	527
p88k	A B	33,406 41,079	123	8 10	125	4 7	175

Experimental Results

circ.	Atalanta fs	Atalanta no fs	PASSAT Eqn	PASSAT SAT
c6288	0,75	49,43	4,78	1,79
c7552	5,72	65,93	2,61	0,70

Time to classify faults

circuit	Time for classification				
	<0.1	0.1-1	1-10	10-20	abort
P44k	0	57	19	0	0
P49k	0	0	385	0	1581
P80k	9	207	0	0	0
P88k	106	167	7	0	0
P177k	137	119	58	5	13
P565k	961	440	8	0	0

Multi-valued model: Industrial Circuits

Circuit	Cnt	Red	Ab	Eqn (s)	SAT (s)
P44k	61230	823	0	17821	30797
P77k	126338	0	0	1156	334
P80k	176159	5	9	7420	5591
P88k	126929	2354	169	2985	9044
P99k	131913	759	4548	4364	36965
P565k	1175605	26372	28343	1456	3073

Challenges / Future Work

- Use of advanced SAT techniques
  - incremental SAT
- Optimization of SAT instance
  - Boolean reasoning during creation
- Other fault models
  - dynamic model, e.g. path delay faults

Conclusions

- SAT for ATPG
- Formulation based on formal techniques
- Use of structural information
- Advanced SAT techniques
- Multi-valued circuits

- Better run times for “hard” faults
- Applicable to large industrial circuits