Gabriel Soares Porto

Uma Análise sobre a eficiência de Geradores Automáticos de Padrões de Teste Híbridos

Brasil

2018

Gabriel Soares Porto

Uma Análise sobre a eficiência de Geradores Automáticos de Padrões de Teste Híbridos

Dissertação de Mestrado

Universidade Federal do Rio Grande (FURG)

Centro de Ciências Computacionais

Programa de Pós Graduação em Computação (PPGComp)

Orientadores: Prof. Dr. Denis Teixeira Franco

Prof. Dr. Paulo Francisco Butzen

Brasil

2018

Ficha catalográfica

P853a Porto, Gabriel Soares.
Uma análise sobre a eficiência de geradores automáticos de padrões de teste híbridos / Gabriel Soares Porto. – 2018.
145 f.

Dissertação (mestrado) – Universidade Federal do Rio Grande – FURG, Programa de Pós-Graduação em Computação, Rio Grande/RS, 2018.
Orientador: Dr. Denis Teixeira Franco.
Coorientador: Dr. Paulo Francisco Butzen.

1. Microeletrônica 2. Ferramentas de EDA 3. Geração Automática de Padrões de Teste 4. ATPG 5. SAT 6. Métodos Estruturais I. Franco, Denis Teixeira II. Butzen, Paulo Francisco III. Título.

CDU 621.3.049.77

Catalogação na Fonte: Bibliotecário José Paulo dos Santos CRB 10/2344

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO RIO GRANDE
CENTRO DE CIÊNCIAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO
CURSO DE MESTRADO EM ENGENHARIA DE COMPUTAÇÃO

DISSERTAÇÃO DE MESTRADO

Uma análise sobre a eficiência de ATPGs Híbridos

Gabriel Soares Porto

Banca examinadora:

Prof. Dr. Tiago Roberto Balen

Prof. Dr. Odorico Machado Mendizabal

Prof. Dr. Denis Teixeira Franco
Orientador

Prof. Dr. Paulo Francisco Butzen
Coorientador

Resumo

O processo de teste de circuitos integrados tem grande importância para detectar possíveis erros gerados por sistemas digitais, especialmente quando se trata de aplicações críticas como em sistemas médicos e aeroespaciais. Ao longo dos anos, o avanço tecnológico inspirado pela Lei de Moore contribuiu para que a miniaturização afetasse diretamente a densidade dos circuitos integrados, bem como sua complexidade, tornando-os mais propensos a falhas e mais difíceis de serem testados. Para atender essa demanda, foram propostos meios para facilitar o teste, tais como técnicas de projeto para Testabilidade e ferramentas como ATPG (*Automatic Test Pattern Generation*), que é o foco deste trabalho.

Ferramentas ATPG têm como objetivo gerar um conjunto reduzido de padrões de teste que possuam uma alta taxa de cobertura de falhas em tempo de execução hábil. Atualmente, há dois principais métodos ATPG: métodos estruturais e métodos baseados em SAT (Método de Satisfatibilidade Booleana). Os métodos estruturais executam uma análise topológica sobre o circuito, ao passo que métodos baseados em SAT apresentam uma abordagem algébrica e funcional, analisando o circuito por meio de expressões booleanas. Os métodos estruturais são ótimos na geração de padrões de teste para falhas fáceis de testar (*easy-to-test faults*); por outro lado, os métodos baseados em SAT são bons na geração de padrões de teste para falhas difíceis de testar (*hard-to-test faults*).

Métodos híbridos buscam a composição e a interação entre variados métodos ATPG. Desse modo, o presente trabalho tem por objetivo dissertar sobre a área de teste e a implementação e análise de métodos ATPG híbridos, com o intuito de difundir mais a relevância das mesmas. Levando em consideração as características de cada método e uma boa estruturação, os resultados deste trabalho apresentaram metodologias híbridas que conseguem reduzir o tempo de execução em até 90%, mantendo um equilíbrio em outros fatores, tais como cobertura de falhas e conjunto de padrões de teste.

Palavras-chaves: Microeletrônica. Ferramentas de EDA. Geração Automática de Padrões de Teste. ATPG. SAT. Métodos Estruturais.

Abstract

The integrated circuits testing process has great importance to detect errors produced by digital systems, especially in critical applications such as medical and aerospace systems. Over the years, the technological advances inspired by Moore's Law made the scaling affects directly the integrated circuit density and its complexity, making them fault prone and more difficult to test. As a result, ways to facilitate the testing process techniques like Design for Testability techniques and softwares like ATPG (Automatic Test Pattern Generation) have been proposed.

ATPG tools aim to generate a reduced test pattern set that guarantees a high fault coverage and an acceptable testing process execution time. There are two main ATPG approaches: structural and SAT-based (Boolean Satisfiability Method). Structural methods perform a topological analysis, and SAT-based methods are based on algebraic and functional approaches to deal with the Boolean expression as the circuit structure. Structural methods are great for generating test patterns for easy-to-test faults; on the other hand, SAT-based methods are good for hard-to-test faults.

Hybrid ATPG methods seek for the composition and the interaction between different ATPG methods. The goals of this work are to discourse about testing process, the implementation and analysis of different hybrid ATPGs methods, focusing on the pros and cons of their use. The results of this work show that, taking into account the characteristics of each method and a good structuring in the interaction between them, hybrid approaches can reduce the execution time by up to 90%, generating a pattern set and fault coverage similar to the standalone version methods.

Key-words: Microelectronic. EDA Tools. Automatic Test Pattern Generation. ATPG. SAT. Structural Methods.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ATPG	Geração Automática de Padrões de Teste (*Automatic Test Pattern Generation*)
BIST	Auto-teste Integrado (*Built-in self-test*)
CI	Circuito Integrado
CNF	Forma Normal Conjuntiva (*Conjunctive Normal Form*)
CUT	Circuito Sob Test (*Circuit Under Test*)
DALG	Algoritmo-D (*D-Algorithm*)
DFT	Projetar para Testabilidade (*Design for Testability*)
ECAT	Correção de Erro e Tradução (*Error-correction-and-translation*)
EDA	Automação de Projeto Eletrônico (*Electronic Design Automation*)
FAN	Algoritmo de Geração de Teste Orientado à *Fan-out* (*Fan-out Oriented Test Generation Algorithm*)
FoM	Figura de Mérito (*Figure of Merit*)
GPU	Unidade de Processamento Gráfico (*Graphics Processing Unit*)
MFFC	Cone Máximo Livre de Fan-out (*Maximal Fan-out Free Cone*)
PDC	Propagação do Cubo-D (*Progation D-Cube*)
PDCF	Cubo-D Primitivo para uma falha (*Primitive D-Cube for a fault*)
PI	Entrada Primária (*Primary Input*)
PO	Saída Primária (*Primary Output*)
POS	Produto de Somas (*Product of Sums*)
PODEM	*Path-Oriented Decision Making*
SA	*Stuck-at*
SAT	Problema de Satisfabilidade Booleana
SC	Cobertura Singular (*Singular Cover*)
TC	Cubo de Teste (*Test Cube*)

Sumário

1 Introdução

Ao longo dos anos, o avanço tecnológico inspirado na Lei de Moore promoveu a miniaturização e, consequentemente, o aumento de densidade de transistores em uma mesma área fez com que a complexidade do projeto de circuitos integrados (CI) aumentasse drasticamente. Com o aumento da complexidade, metodologias de síntese de projeto como a *Standart Cell*, aliadas a ferramentas de síntese e biblioteca de células, se tornaram essenciais para a automatização no desenvolvimento de sistemas digitais modernos.

Após serem fabricados, circuitos integrados são submetidos a etapa de teste. O processo de teste de circuitos integrados sempre foi de suma importância afim de verificar se o sistema fabricado possui algum defeito, principalmente no tocante a sistemas críticos, como em aplicações médicas e aeroespaciais.

O processo tradicional de teste de circuitos integrados consiste na aplicação de todos os possíveis estímulos que um circuito possa receber, afim de verificar possíveis erros nas saídas provenientes de defeitos. No entanto, realizar o teste tradicional exaustivo se tornou inviável com o aumento da complexidade dos sistemas e, consequentemente, meios para facilitação do processo de teste se tornaram essenciais.

Diversos meios para o auxílio no processo de teste começaram a surgir, tais como técnicas de projetar para testabilidade (DFT ou *Design for Testability* do inglês) que visam a facilitar o teste de circuitos adicionando estruturas ao seu projeto. Outro meio de auxílio para o teste – sendo esse o foco do presente trabalho – consiste nas ferramentas conhecidas por ATPG (*Automatic Test Pattern Generation*). Ferramentas ATPG têm por objetivo a geração, de forma automática, de um conjunto reduzido de padrões de teste que gere alta taxa de detecção de falhas ou cobertura de falhas. Esse tipo de ferramenta busca otimizar o processo de teste, possibilitando testar o circuito integrado em tempo hábil.

Atualmente, ferramentas ATPG podem ser divididas em duas principais classes: as estruturais e as baseadas em SAT (Método de Satisfatibilidade Booleana). Ferramentas estruturais são aquelas baseadas nos algoritmos clássicos, como o Algoritmo-D (ROTH, 1966), PODEM (GOEL, 1981) e FAN (FUJIWARA; SHIMONO, 1983). Esse tipo de ferramenta trata o circuito de forma topológica e seu desempenho é muito dependente do modelo de falhas utilizado. Como a abordagem está ligada à topologia do circuito, atualmente esse tipo de metodologia acaba por sofrer problemas de escalabilidade, devido ao aumento no número de funções lógicas dos circuitos modernos. A alternativa encontrada para contornar o problema foi a aplicação de resolvedores SAT em ferramentas ATPG, tratando o circuito com uma abordagem algébrica e funcional. Ferramentas SAT estão

ligadas à avaliação da expressão Booleana do circuito, fazendo com que sejam mais flexíveis no que diz respeito à modelagem de falhas e mais eficientes na geração de padrões (CHEN; MARQUES-SILVA, 2013; LARRABEE, 1992).

Quando ferramentas SAT aplicadas a ATPG foram propostas em (LARRABEE, 1992) não havia maturidade suficiente, naquele contexto, para substituir ferramentas industriais consolidadas as quais utilizavam soluções estruturais. No entanto, atualmente várias ferramentas eficientes como Chaff (MOSKEWICZ et al., 2001), GRASP (MARQUES-SILVA; SAKALLAH, 1999) e MINISAT (EéN; SöRENSSON, 2004) garantem maiores praticidade e desempenho em relação às soluções de ATPG estruturais. Tais vantagens se sobressaíram quando, com a evolução de CIs, advieram problemas de escalabilidades devido à metodologia estrutural aplicada. Atualmente, resolvedores SAT apresentam abordagens eficientes que atendem a demanda atual. No entanto, com o avanço da tecnologia a busca por aprimoramentos em ATPG são constantes.

De modo geral, métodos estruturais têm uma maior eficiência para o processamento de falhas de fácil detecção (*easy-to-test faults* do inglês), ao passo que resolvedores SAT obtêm um resultado melhor para falhas de difícil detecção (*hard-to-test faults* do inglês) (DRECHSLER et al., 2008). A combinação de métodos ATPG de diferentes características visa um melhor aproveitamento do potencial de cada abordagem, afim de aumentar o desempenho de aplicações voltadas a geração de padrões de teste. Desta forma, atualmente estão sendo investigados métodos híbridos, combinando estratégias estruturais e baseadas em SAT (CHEN; MARQUES-SILVA, 2013).

1.1 Objetivo do Trabalho

Este trabalho acadêmico tem por objetivo dissertar sobre teste de circuitos integrados e seus principais conceitos, realizar a implementação de uma ferramenta ATPG contendo métodos em suas versões individuais e híbridas, bem como proceder uma análise acerca da eficiência de ATPGs híbridos, a fim de demonstrar a relevância da utilização da composição de métodos ATPG.

1.2 Organização do Texto

O trabalho está organizado da seguinte maneira. O Capítulo 2 apresenta a fundamentação teórica na qual este trabalho está embasado, bem como o estado-da-arte. O Capítulo 4 introduz a estrutura e o funcionamento da ferramenta desenvolvida para obtenção do objetivo final: a análise de métodos híbridos. O Capítulo 5 expõe as análises realizadas referentes à eficiência de métodos híbridos. Por fim, no Capítulo 6 é apresentado as conclusões obtidas por meio de toda a revisão literária e análises realizadas.

2 Referencial Teórico

Neste capítulo, serão expostos o embasamento teórico pertinente ao entendimento do trabalho desenvolvido e também o estado-da-arte referente aos métodos ATPG. Na Seção 2.1 são abordados conceitos importantes de defeito, falha, erro e falha sistêmica. Na Seção 2.2 são apresentados conceitos como o de modelo de falhas, observabilidade e controlabilidade, terminologias de falhas e o método *fault collapsing*. Na Seção 2.3 é feita uma breve introdução sobre *Design for Testability*, princípios essenciais utilizados em ATPGs estruturais e o método clássico Algoritmo-D. Na seção 2.4 será introduzido o método de satisfatibilidade booleana e o modo como é projetado para métodos de geração de padrões. Por fim, na Seção 3 é apresentado o estado-da-arte em ATPG.

2.1 Conceitos de Defeito, Falha, Erro e Falha Sistêmica

Neste trabalho são baseados nos conceitos utilizados em (WANG; CHANG; CHENG, 2006) para defeito, falha, erro e falha sistêmica. O conceito de defeito está ligado ao universo físico, já falha e erro relacionam-se ao universo da informação e, por fim, o conceito de falha sistêmica se refere ao universo do usuário, conforme sintetizado na Figura 1.

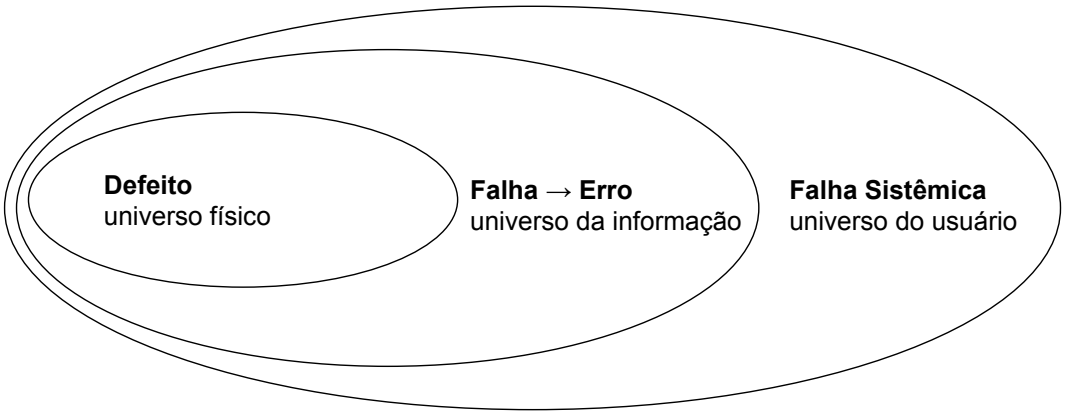


Figura 1 – Adaptação gráfica dos conceitos de defeito, falha, erro e falha sistêmica de (WANG; CHANG; CHENG, 2006).

Primeiramente, defeito é uma imperfeição física que pode acarretar em uma falha. O defeito pode ser oriundo de imperfeições vindas de falhas de projeto, pelo processo de manufatura, pela utilização do sistema digital ou distúrbios como ruídos e radiação. O conceito de falha refere-se a representação de um defeito, refletindo em uma condição física que leva o circuito a manifestar um estado errôneo em certas condições. Quando certas condições são atendidas em um circuito com falha, e estas levam saídas do circuito a um

estado diferente do esperado é dito que houve um erro. Todavia para o erro, dependendo de sua magnitude, pode se manifestar como uma falha sistêmica, ou seja, um desvio do comportamento especificado para o sistema, o que acarreta repercussões no universo do usuário. A falha sistêmica é irreversível, haja vista que esse tipo de comportamento não pode ser tolerado, apenas evitado. De modo geral, um circuito que apresente um defeito pode ser levado a apresentar uma falha. A falha, por sua vez, pode provocar erros no circuito e, por fim, um circuito errôneo pode acarretar uma falha sistêmica, refletindo esse estado ao usuário.

2.2 Teste de Circuitos

Quando uma falha leva a um erro tem-se um estado irreversível, o que deve ser evitado. Dessa maneira, o objetivo do processo de teste de circuitos é a detecção de erros para verificar se o sistema fabricado está de acordo com as especificações de projeto, sobretudo em sistemas críticos como aplicações médicas e aeroespaciais.

De modo geral, para testar um circuito com *n+1* entradas e *m+1* saídas primárias, um conjunto de padrões é aplicado a um circuito sob teste (*circuit under test* ou CUT do inglês) e as respostas de suas saídas são comparadas com as respostas conhecidas de um circuito livre de falhas (ou *fault-free circuit* do inglês), como ilustrado na Figura 2. Caso a comparação resulte em valores diferentes, um erro decorrente a falha no circuito é detectado no CUT.

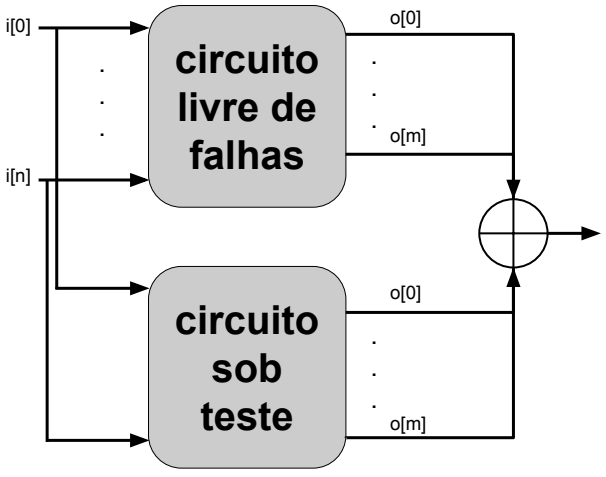


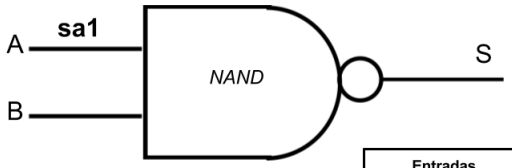
Figura 2 – Esquematização do processo de teste de circuitos.

2.2.1 Modelo de Falhas

Um importante fator no processo de teste é o modelo de falha empregado, o qual pode ser o determinante na qualidade final do processo de teste. O modelo de falha

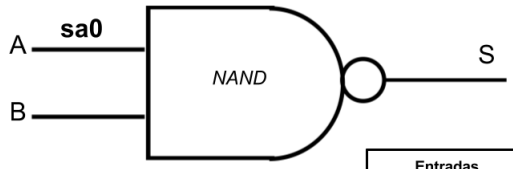
utilizado neste trabalho é conhecido por falha de colagem simples (do inglês *stuck-at fault*).

O modelo de falha de colagem ou *stuck-at* é aquele que força um valor lógico fixo – zero ou um – a uma entrada ou saída de uma porta lógica ou em um *flip-flop* do circuito (BUSHNELL; AGRAWAL, 2000). Esse modelo de falha pode ser dividido em dois tipos: *stuck-at-1* ou *sa1* para as que forçam o valor lógico "1" (visto na Figura 3a) e *stuck-at-0* ou *sa0* para o valor lógico "0" (visto na Figura 3b). Um circuito com n sinais possui $2n$ falhas *stuck-at*.



Entradas		Saídas	
A	B	S	S (falha)
0	0	1	1
0	1	1	0
1	0	1	1
1	1	0	0

(a) Exemplo de Falha *Stuck-at-1* em uma porta *NAND*



Entradas		Saídas	
A	B	S	S (falha)
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	1

(b) Exemplo de Falha *Stuck-at-0* em uma porta *NAND*

Figura 3 – Exemplos de Falhas *Stuck-at*

O efeito da falha *stuck-at-1* pode ser observado no exemplo da Figura 3a, onde foi aplicada uma falha *sa1* na entrada A de uma porta NAND de duas entradas (A e B) e saída S. Pode-se observar que em apenas um dos quatro possíveis casos de padrões de entrada o efeito da falha se manifesta na saída. Para o padrão "01" o efeito da falha é propagado até a saída, onde o valor esperado seria "1", mas passa a ser "0" na presença da falha. Para os demais padrões "00", "10" e "11" o efeito de falha é mascarado, ou seja, mesmo na presença de falha a saída do circuito possui o valor esperado. Análise complementar é ilustrada na Figura 3b, para a falha do tipo *stuck-at-0*.

Ferramentas de geração de padrões de teste tratam falhas do tipo *stuck-at* única, ou seja, considerando apenas uma falha por vez no circuito. Um adequado conjunto de

padrões de teste que consiga prover uma boa cobertura de falhas de tipo *stuck-at* também garante uma boa cobertura de múltiplas falhas não somente para *stuck-at*, mas também para outros tipos de falhas (NAVABI, 2011).

2.2.2 Observabilidade e Controlabilidade

No processo de geração de teste, estímulos devem ser inseridos nas entradas do circuito para que o efeito da falha seja observável na saída. Há situações encontradas na etapa de teste em que se deve estimular valores de entradas e de sinais internos do circuito para que se possa verificar a funcionalidade do circuito. Nessas condições, é necessário saber o quão difícil é controlar valores em alguns nodos do circuito por meio das entradas primárias ou o quão observáveis são alguns valores em nodos internos do circuito, afetando diretamente a geração de teste (NAVABI, 2011).

A definição de controlabilidade é dada como a medida da dificuldade de atribuir um certo valor a um certo nodo de um circuito (NAVABI, 2011). Tenha como exemplo o circuito da Figura 4. Para encontrar um teste para a falha de colagem simples, *stuck-at-0*, é necessário ter o valor 1 no sinal em questão marcado com um *"x"* em vermelho. Para possibilitar o controle sobre o sinal que foi aplicado à falha, é necessário aplicar o valor de controle 0 em uma das entradas, sinais I_1 ou I_3, da porta NAND. Com esse exemplo, é possível observar que os sinais com maior controlabilidade são as entradas primárias (PI, do inglês *Primary Inputs*) do circuito e, quanto mais interno o nodo, mais difícil é seu controle, já que valores mais internos necessitam de mais valores atribuídos a sinais do circuito para serem controlados.

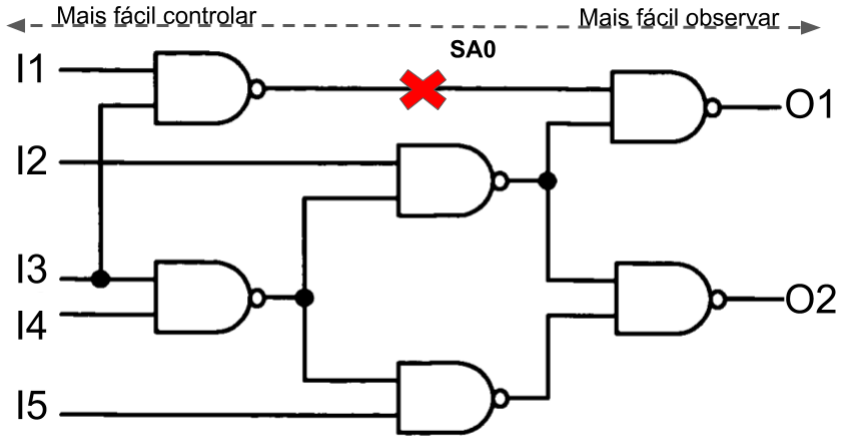


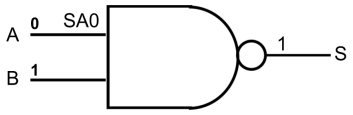
Figura 4 – Exemplo de conceitos de Controlabilidade e Observabilidade no Circuito ISCAS85 C17 (BRYAN, 1985)

O conceito de observabilidade é definido como a medida da dificuldade de observação da mudança de um valor em uma das saídas primárias (PO, *Primary Outputs*) do circuito (NAVABI, 2011). No exemplo da Figura 4, o efeito da falha aplicada no sinal

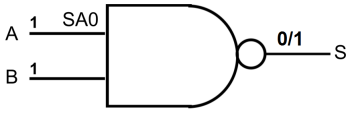
assinalado com o *"x"* vermelho será observável se a falha for ativada e a outra entrada da porta NAND for controlada proporcionando a propagação do efeito da falha até a saída primária do circuito. Pode-se concluir que, enquanto nas entradas primárias a controlabilidade é maior, a observabilidade é maior nas saídas primárias.

2.2.3 Terminologias de Falhas

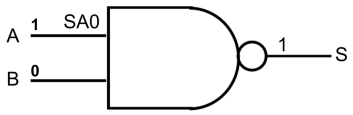
Para uma hábil compreensão do texto, serão abordadas várias definições importantes a seguir. Visando a uma melhor consistência, foram utilizados os conceitos adotados por (NAVABI, 2011) e adaptados baseados nos conceitos descritos na Seção 2.1. A Figura 5 ilustra exemplos das terminologias que serão apresentadas.



(a) Falha SA0 no sinal A não ativada



(b) Propagação da falha SA0 no sinal A para a saída da porta lógica NAND



(c) Mascaramento ou bloqueamento da falha SA0 no sinal A

Figura 5 – Exemplos Gráficos das Terminologias sobre Falhas

Uma falha é dita ativada se, e somente se, o valor lógico original no sinal afetado por ela diferir do valor falho. Por exemplo, para uma falha do tipo *stuck-at-1*, a qual fixa o valor lógico "1", para ser considerada ativada é necessário o valor "0" no sinal original. O mesmo acontece para uma falha *stuck-at-0* que, para ser ativada, é necessário o valor lógico "1" no sinal em questão. Um exemplo no qual a falha não é ativada pode ser visualizado na Figura 5a. A falha SA0 ou *stuck-at-0* no sinal A exige o valor lógico "1" no sinal original em questão.

Quando uma falha é ativada, ela pode ser propagada ao longo do circuito. Para essa condição acontecer, a falha aplicada a uma das entradas deve ser ativada, ao passo que nos outros sinais da porta lógica são aplicados valores lógicos que façam com que o valor na saída difira do valor lógico esperado. No exemplo da Figura 5b, uma porta NAND de duas entradas apresenta na entrada A uma falha do tipo SA0. Quando o valor lógico de A for "1", a falha é considerada ativa. Para que o efeito da falha seja propagado, a entrada B também terá que assumir o valor lógico "1". Caso contrário, a falha será mascarada.

Quando é aplicado um valor na entrada da porta lógica que faça com que a falha não se propague é dito que foi bloqueada ou mascarada. A Figura 5c representa um caso de mascaramento de falha, em que a falha SA0 é ativada, porém na outra entrada é aplicado o valor lógico "0", fazendo com que a falha seja mascarada/bloqueada.

Uma falha é detectada se, dado um padrão de teste, ela é ativada e propagada até alcançar uma das saídas primárias gerando um erro na saída do circuito. Dado um padrão de teste, se a falha em questão for mascarada, então a mesma não é detectada e, consequentemente, tal padrão de teste não contém relevância para o processo. É buscado, então, que o conjunto de padrões de teste gerado por meio do método ATPG consiga detectar o maior número de falhas possível sem comprometer a qualidade de teste.

Dado um conjunto de padrões de teste, a cobertura de falhas é definida pela razão entre a quantidade de falhas detectadas (utilizando este conjunto de teste) e o total de possíveis falhas contidas no circuito (NAVABI, 2011). O referido conceito pode ser descrito pela Equação 2.1.

$$Cobertura\ de\ Falhas = \frac{N^o\ de\ Falhas\ Detectadas}{N^o\ Total\ de\ Falhas} \tag{2.1}$$

Cobertura de falhas é uma métrica largamente utilizada para avaliar a qualidade do conjunto de padrões de teste gerado pelo ATPG. Contudo, é praticamente impossível obter 100% de cobertura de falhas, uma vez que sempre há parte das falhas contidas em um circuito que não são detectáveis. Desse modo, a métrica na qual consiste a cobertura de falhas pode ser modificada e expressa como eficiência na detecção de falhas ou também cobertura de falhas efetiva (WANG; CHANG; CHENG, 2006). A métrica pode ser reescrita como:

$$CF\ Efetiva = \frac{N^o\ de\ Falhas\ Detectadas}{N^o\ Total\ de\ Falhas - N^o\ de\ Falhas\ Não\ Detectáveis} \tag{2.2}$$

Entretanto, para a utilização da Equação da Cobertura de Falhas Efetiva (2.2) é necessário que todas as falhas não detectáveis sejam identificadas. Mesmo sendo uma métrica mais precisa, a sua utilização torna-se dependente de uma tarefa difícil.

2.2.4 *Fault Collapsing*

Fault Collapsing é um processo utilizado para otimização de ATPGs, que visa reduzir o tamanho do conjunto de falhas a serem simuladas e analisadas. Ele possui como objetivo a redução de esforços para etapas como síntese e análise de padrões de teste, sem reduzir a qualidade dos testes (UBAR et al., 2010). A aplicação de tal método é baseada em relações entre falhas, que podem se dar de duas formas: equivalência e dominância.

A primeira relação, conhecida como equivalência, é definida quando duas ou mais falhas são detectadas unicamente pelo mesmo conjunto de padrões de teste; assim, apenas uma falha desse conjunto é considerada para ser analisada posteriormente na etapa de síntese de padrões de teste. Dessa maneira, pode-se definir que para cada padrão de teste existe um conjunto de falhas proveniente da aplicação de relações entre as possíveis falhas do circuito. A relação de equivalência pode ser definida por meio da Equação 2.3, ou pela sua forma reduzida definida pela Equação 2.4.

$$[F_0(V) \oplus F_1(V)] \oplus [F_0(V) \oplus F_2(V)] = 0 \tag{2.3}$$

$$F_1(V) \oplus F_2(V) = 0 \tag{2.4}$$

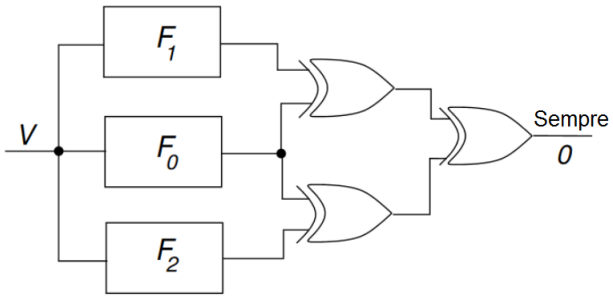
A Equação 2.3 representa a igualdade ou equivalência na divergência de resultados das funções F_1 e F_2 em comparação à função F_0, onde a função geral F representa a expressão lógica do circuito em questão, F_0 define o circuito livre de falhas e F_1 e F_2 representa o circuito lógico na presença de duas falhas diferentes: (falha f_1 e f_2). Quando um mesmo padrão de entrada V é aplicado às três funções, F_0, F_1 e F_2, são aplicadas operações de disjunção exclusiva a fim de comparar se houve ou não divergência nos resultados das funções.

As falhas aplicadas, f_1 e f_2, às funções F_1 e F_2, respectivamente, serão equivalentes se, e somente se, para todos os padrões de entrada as Equações 2.3 ou 2.4 resultem sempre em "0" lógico. As Equações 2.3 e 2.4 podem ser representadas graficamente, para um melhor entendimento, pelos circuitos apresentados na Figura 6.

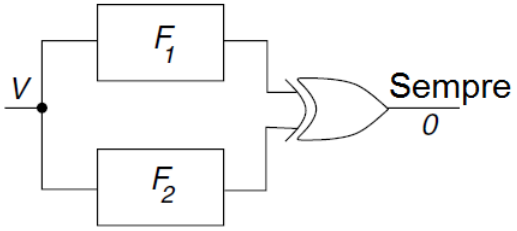
A segunda relação, conhecida por dominância, é definida como: todo o conjunto de padrões aplicados à função F_1 que detecte a falha f_1 é um subconjunto de padrões que também detectam a falha f_2 presente na função F_2. É dito, então, que a falha f_2 domina f_1. Desse modo, pode ser desconsiderada a falha f_2 aplicada à F_2, reduzindo o conjunto de falhas final. É possível definir a relação de equivalência através de uma dominância bi-implicativa, ou seja, se a falha f_1 domina f_2 e a falha f_2 domina f_1 uma relação de equivalência pode ser identificada.

Na condição em que f_1 domina f_2, qualquer padrão que detecte a falha f_2 aplicado à função F_2 deve detectar também a falha f_1 aplicada à função F_1. Pela regra da contra-positiva, qualquer padrão que não detecte a falha $f1$ aplicado à função F_1, não deve detectar a falha $f2$ aplicada à função F_2 (AGRAWAL; PRASAD; ATRE, 2003). Assim sendo, é possível definir a relação de dominância pela Equação 2.5, bem como sua representação gráfica pode ser vista na Figura 7.

$$[F_0(V) \oplus F_2(V)]\,\overline{[F_0(V) \oplus F_1(V)]} = 0 \tag{2.5}$$



(a) Representação Gráfica da Equação 2.3



(b) Representação Gráfica da Equação 2.4

Figura 6 – Representação da Relação de Equivalência entre Falhas. Adaptado de (SANDIREDDY; AGRAWAL, 2005)

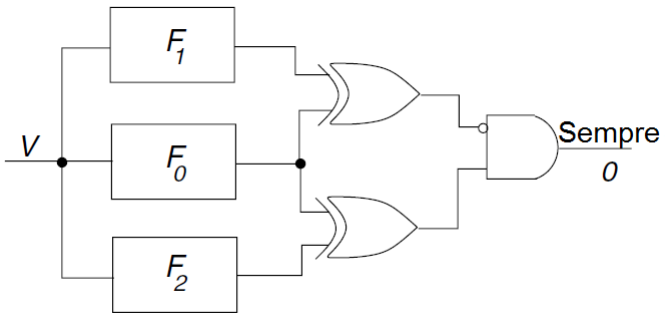


Figura 7 – Representação Gráfica da Equação 2.5. Adaptado de (SANDIREDDY; AGRAWAL, 2005)

Existem duas metodologias para a aplicação de relações entre falhas: a metodologia Estrutural e a Funcional. Os Métodos Estruturais realizam uma análise topológica do circuito (LIOY, 1991), ou seja, as relações entre falhas são aplicadas apenas entre entradas e saídas de portas lógicas. Para cada tipo de porta lógica existe um determinado padrão de falhas a ser considerado e aplicado ao longo do circuito. Métodos Estruturais apresentam uma redução do conjunto de falhas a serem simuladas e analisadas por volta dos 50%, de acordo com a literatura (AGRAWAL; PRASAD; ATRE, 2003).

Uma maior redução do conjunto de falhas é possível com os métodos de análise Funcional. Esses levam em conta todo o circuito para aplicar as relações, ou seja, é avaliada a mudança realizada por cada falha na expressão Booleana final do circuito. Por

exemplo, para um Método de Equivalência Funcional, a relação é observada quando duas falhas diferentes (independentemente de onde são aplicadas no circuito), modificam a expressão Booleana do circuito da mesma maneira, ou seja, produzem os mesmos valores lógicos na saída. Devido ao aumento da complexidade dos circuitos, sua aplicabilidade se tornou impraticável. Posteriormente, ATPGs começaram a utilizar esse conceito de análise funcional para a geração de padrões de teste, o que culminou em um aumento de desempenho e praticidade, conseguindo lidar com problemas de escalabilidade.

2.3 Geração de Teste

Ao longo dos anos, com a miniaturização dos componentes de circuitos integrados guiados pela Lei de Moore, os projetos tornaram-se cada vez mais complexos e densos. Consequentemente, o processo de teste se tornou impraticável de ser feito exaustivamente e, assim, meios para geração de testes que garantissem uma boa cobertura de falhas começaram a surgir para suprir tal necessidade. Para lidar com a demanda, métodos como algoritmos de geração automática de padrões de teste e técnicas de projeto para Testabilidade para aumento de controlabilidade e observabilidade como *Scan Design* e BIST (*built-in self-test*) foram propostos (WESTE; HARRIS, 2011).

2.3.1 *Design for Testability*

A chave para desenvolver circuitos que obtenham uma ótima testabilidade consiste na controlabilidade e na observabilidade, sabendo que testabilidade pode ser definida como a medida relativa do esforço ou custo empregado no processo de teste de um circuito lógico, logo, quanto mais controlável e observável é um circuito, maior é seu grau de testabilidade (WANG; WU; WEN, 2009). Podemos concluir que, em um circuito com boa controlabilidade e observabilidade, o custo do processo de teste é diminuído, além de se prover uma alta cobertura de falhas (WESTE; HARRIS, 2011).

A seguir, será feita uma breve introdução de duas técnicas de DFT (*Design for Testability*) que visam a auxiliar na geração de padrões de teste. São elas: *Scan Design* e *built-in self-test* ou BIST.

A estratégia de *Scan Design* para teste consegue prover um aumento de controlabilidade e observabilidade. O objetivo é facilitar o teste, acessando cadeias de registradores no circuito. Em projetos modernos, são utilizados registradores *scan*, um *flip-flop D* precedido de um multiplexador, como mostrado na Figura 8.

Quando o modo *scan* está desligado, o comportamento dos registradores é o padrão, armazenando dados nos *flip-flops*. Quando o modo *scan* é ligado, os dados desejados são carregados pelo sinais de *scan-in*, possibilitando testar padrões desejados e verificar saídas a partir das cadeias. Geração de teste utilizando esse tipo de arquitetura pode ser

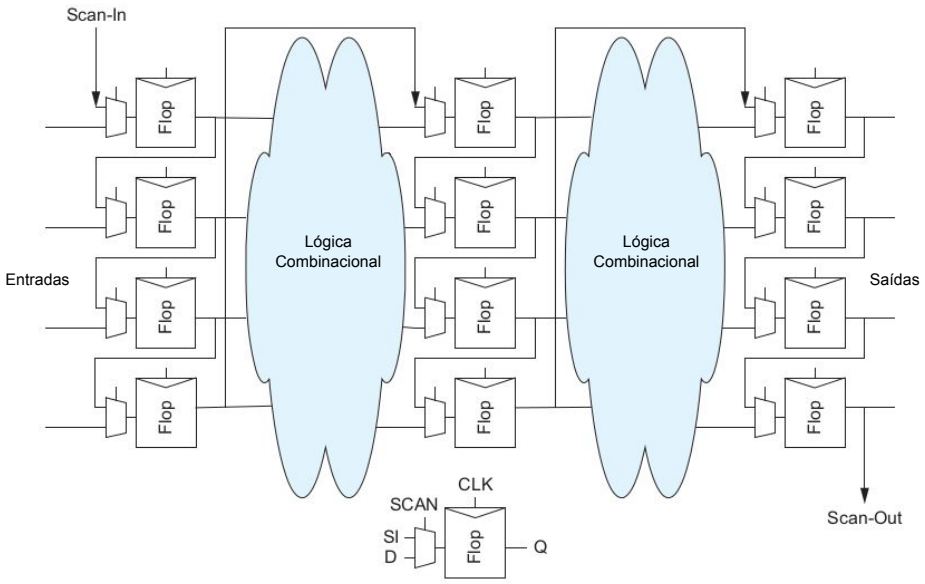


Figura 8 – Teste baseado em cadeias *Scan*. Adaptado de (WESTE; HARRIS, 2011).

altamente automatizada. Métodos ATPG podem ser aplicados aos blocos lógicos, podendo ser feita a análise por meio das cadeias *scan*. Porém, as maiores desvantagens dessa técnica são o aumento de área e o atraso, devido aos registradores (WESTE; HARRIS, 2011).

A técnica de DFT (*Design for Testability*) BIST ou *built-in self-test* – de acordo com o próprio nome – reside em aumentar a estrutura de circuitos, permitindo que sejam feitas operações sobre eles mesmos, e que consigam aferir se seu funcionamento está correto. Essas técnicas adicionam área ao circuito, entretanto, reduzem o tempo de teste necessário e podem diminuir o custo geral do sistema digital (WESTE; HARRIS, 2011).

O funcionamento típico de um sistema com a técnica BIST pode ser visto na Figura 9. O sistema é composto basicamente por um Gerador Automático de Padrões de Teste (TPG ou *Test Pattern Generator* do inglês), responsável pela geração de padrões que irão ser aplicados no circuito sob teste. A resposta do circuito, então, é passada ao Analisador de Saídas (ORA ou *Output Response Analyzer*), o qual irá analisar a saída do circuito com o resultado esperado, de acordo com sua função lógica. Por sua vez, o Controlador BIST irá coordenar as operações realizadas entre os três módulos citados acima, por meio de sinais de controle (WANG; CHANG; CHENG, 2006).

2.3.2 ATPG: Geração Automática de Padrões de Teste

O processo de geração automática de padrões de teste é o meio de automatização do processo de teste abordado no desenvolvimento deste trabalho. Os processos de automatização de testes apresentados anteriormente experimentam adicionar estruturas dentro do circuito visando aumentar a controlabilidade e observabilidade para facilitar o teste (tal como é a técnica *Scan Design*) e meios de auto-teste utilizando módulos de *hard-*

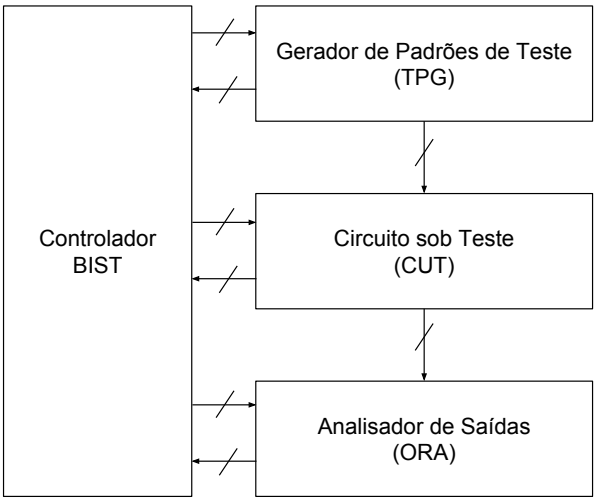


Figura 9 – Exemplo de um sistema BIST típico. Adaptado de (WANG; CHANG; CHENG, 2006).

ware adicional (como é a técnica BIST). Diferentemente desses dois últimos, os ATPGs utilizam *softwares* para análise e síntese de um conjunto reduzido de padrões de teste que garantam alta cobertura de falhas.

O problema de geração de padrões de teste é o procedimento de encontrar um padrão de teste no qual as saídas primárias do circuito livre de falha e do circuito sob presença de falha f divergem. Quando o processo encontra um padrão é dito que se detectou a falha f em questão. Entretanto, há alguns casos em que o comportamento do circuito livre de falhas e do circuito sob presença de falhas é o mesmo. Falhas que não afetam de forma alguma as saídas do circuito são chamadas de falhas redundantes.

Atualmente, ATPGs podem ser divididos em duas grandes classes: métodos estruturais e métodos baseados em SAT (LARRABEE, 1992). A seguir, serão introduzidos os conceitos básicos de ATPGs estruturais, como seu procedimento clássico, o Algoritmo-D. Subsequentemente, será apresentado o Método de Satisfatibilidade Booleana (SAT), que como pode ser modelado para a obtenção de padrões de teste.

Métodos Estruturais se caracterizam por sua análise topológica do circuito lógico. Desse modo, os algoritmos clássicos (ROTH, 1966; GOEL, 1981; FUJIWARA; SHIMONO, 1983) e subsequentes implementações (SCHULZ; TRISCHLER; SARFERT, 1988; HAMZAOGLU; PATEL, 1998; GIZDARSKI; FUJIWARA, 2002) compartilham dos mesmos conceitos. A seguir serão abordados os principais conceitos utilizados em metodologias estruturais.

O objetivo principal de uma aplicação de geração de padrões de teste é propagar o erro proveniente da falha até uma saída primária, ou seja, torná-lo detectável dado um certo padrão de entrada. Para esse procedimento, levando em conta a análise topológica que métodos estruturais utilizam, é utilizada a definição da notação D para facilitar a

análise.

O conjunto da notação D se resume em um conjunto de cinco valores $\{0, 1, D, \overline{D}, X\}$, representados na Tabela 1. Os valores do conjunto resultam na composição de valores lógicos v/v_f, os quais representam respectivamente o valor do circuito sem falha (*fault-free circuit*) e do circuito com a falha (*faulty circuit*) (PINTO; REIS, 2002).

Para representação do efeito de uma falha no nível topológico é necessário que o valor atribuído a um certo sinal demonstre discrepância. Para isso, são utilizados os valores D e $\overline{D}$ para representar falhas do tipo *stuck-at-0* e *stuck-at-1*, respectivamente. Como visto na Tabela 1, o valor D representa a composição dos valores lógicos "1", valor esperado para circuito sem falha, e "0" para circuito com falha. De forma semelhante, o valor $\overline{D}$ representa a composição dos valores lógicos "0", valor esperado para circuito sem falha, e "1" para circuito com falha. Por fim, o valor restante definido por X, representa um valor desconhecido ou que não possui relevância, podendo assumir tanto o valor "0" como o "1".

Tabela 1 – Notação D. Adaptação de (NAVABI, 2011).

Composição	Definição	Notação D
0/0	0 Bom / 0 Falho	0
1/1	1 Bom / 1 Falho	1
1/0	1 Bom / 0 Falho	D
0/1	0 Bom / 1 Falho	$\overline{D}$
X	*Don't Care*	X

Para que se possa propagar o efeito errôneo e detectar uma falha utilizando notação D, é necessário utilizar o que se pode denominar Álgebra D. A Álgebra D consiste em operações lógicas com variáveis compostas. Por exemplo, o desenvolvimento de uma operação de conjunção lógica entre os valores $\overline{D}$ e 1 resultaria nos seguintes passos: $\overline{D} \cdot 1 = 0/1 \cdot 1/1 = (0 \cdot 1)/(1 \cdot 1) = 0/1 = \overline{D}$.

A etapa de justificação pode ser resumido como o processo de ajuste de valores lógicos de entrada do circuito para ativar falhas ou para facilitar a etapa de propagação através do circuito até uma saída primária (NAVABI, 2011). Considerando uma porta NAND de n entradas (exemplo referente à Figura 10) e levando em conta a sua tabela verdade, é constatado que existe apenas uma combinação de entrada que possibilita a justificação da saída ter o valor "0", Figura 10a. Já para justificar a saída de valor "1" em uma porta NAND é conhecido que existem $2^n - 1$ possibilidades. A maneira mais fácil de justificar a saída "1" é aplicando o valor controlante da porta lógica a uma das entradas, que no caso da porta NAND é "0". Tendo o valor "0" em uma das entradas sempre se obtém a saída "1", independente das demais entradas, ao passo que as demais entradas permanecem indeterminadas com o valor X, como visto na Figura 10b. No caso de ter de

justificar uma falha do tipo *stuck-at-v* na saída da porta, é necessário justificar a saída da porta lógica com o valor $\overline{v}$.

A etapa de propagação basicamente utiliza o conceito de propagação de falhas aplicadas a algoritmos, cujo objetivo é propagar a falha até ela ser detectada em uma das saídas primárias como um erro. Para isso, é necessário encontrar um caminho sensibilizável no circuito, isto é, aquele que vai do sinal onde a falha é considerada até uma das saídas primárias, na qual deve ser obtido um valor D ou $\overline{D}$ e, assim, a falha ser detectada. Basicamente, pode-se descrever o processo pela execução sucessiva de operações de propagação por um certo caminho até alcançar uma PO. Nesse processo são apenas atribuídos valores lógicos de sinais de entrada das portas lógicas pelo caminho que é transcorrido. A etapa de justificação é feita após, para validação do caminho sensibilizável escolhido.

(a) Justificação do valor lógico "0"

(b) Justificação do valor lógico "1"

Figura 10 – Justificação de possíveis saídas para uma porta NAND de 2 entradas

Calculando a tabela da Álgebra D para portas NAND de 2 entradas e utilizando o conceito de propagação e detecção de falhas, obtém-se o resultado exposto na Figura 11. Como exemplo, foi utilizado o circuito mais simples do *benchmark* ISCAS85 (BRYAN, 1985), sendo aplicada uma falha do tipo *stuck-at-0* em um dos ramos do *fan-out* da entrada primária I_3. Utilizando a notação D, podemos representar tal falha pelo valor D. É sabido, por meio da Álgebra D, que em uma porta NAND de 2 entradas, uma falha D sempre irá propagar um erro $\overline{D}$ se a outra entrada estiver com o valor lógico "1", o mesmo para seu complemento, $\overline{D}$ irá propagar D, respeitando à mesma condição quando a outra entrada estiver com o valor "1".

Desse modo, sempre que encontrar um valor D ou $\overline{D}$, aplica-se "1" na outra entrada e será propagado seu complemento. Um problema que podemos encontrar nesse exemplo é a existência de *fan-outs* não unitários. Tal condição pode levar o processo de propagação e justificação a uma inconsistência, que vem a ser a condição de quando é necessário justificar ou inferir um valor lógico a um sinal, porém o valor requerido difere do valor anteriormente já atribuído ao sinal. Inconsistências ocorrem em casos nos quais podemos tomar diferentes escolhas durante o processo de propagação de efeito de falhas e – para esses casos – devem-se realizar etapas de *backtracking*, que consistem em refazer escolhas de caminhos até não se obterem mais inconsistências. A quantidade de *backtracks* define drasticamente o tempo em que o algoritmo estrutural será executado (FUJIWARA; SHIMONO, 1983).

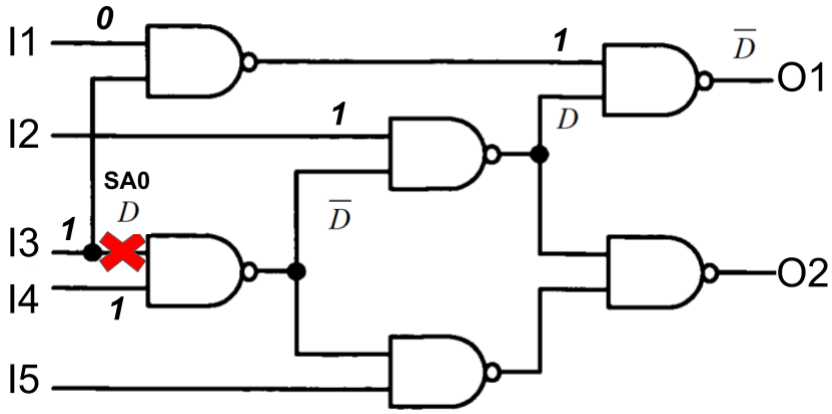


Figura 11 – Exemplo de Propagação de Efeito de Falha no Circuito ISCAS85 C17 (BRYAN, 1985)

2.3.3 Algoritmo-D

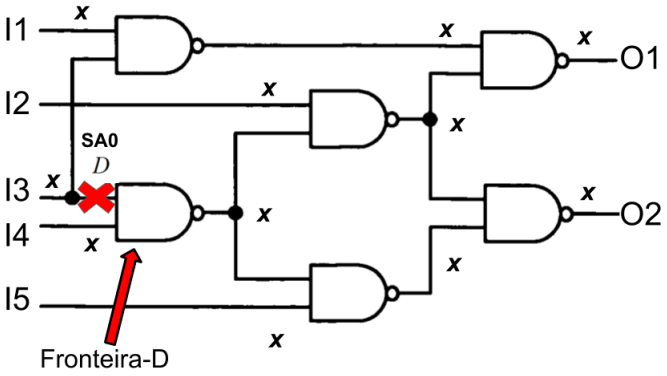
Neste trabalho, será utilizado o método estrutural para geração de padrões Algoritmo D, um método clássico definido em (ROTH, 1966). O método Algoritmo-D para geração de padrões utiliza métodos e conceitos básicos já definidos como propagação e justificação, bem como conceitos introduzidos pelo método que serão abordados a seguir.

Para realizar os processos conforme o Algoritmo-D propõe, são utilizados os conceitos de Fronteira-D e Fronteira-J (*D-Frontier* e *J-Frontier* do inglês), como pode ser visto na Figura 12. A Fronteira-D consiste em uma lista de portas lógicas em que a saída é desconhecida, valor X na saída, e com pelo menos um D ou $\overline{D}$ em uma das entradas, ou seja, é a lista de portas lógicas na qual se pode realizar propagações do efeito da falha. A Figura 12a representa o estado inicial do algoritmo, onde todo circuito é desconhecido, apresentando o valor X, em que apenas em um ponto do circuito é apresentado o efeito de falha D ou $\overline{D}$.

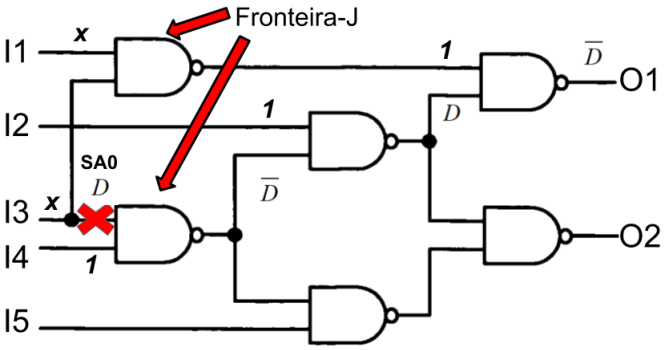
Neste momento, a Fronteira-D é constituída pela porta lógica na qual um dos seus sinais de entrada é exatamente o sinal afetado pela falha aplicada no circuito. A Fronteira-J consiste na lista de portas lógicas que será utilizada na etapa de justificação de sinais, ou seja, onde suas saídas são conhecidas, podendo assumir os valores “0”, “1”, D ou $\overline{D}$, mas suas entradas ainda não foram justificadas. Como visto na Figura 12b, a Fronteira-J possui duas portas em sua lista. A porta NAND que possui as entradas I_1 e I_3 é um exemplo de porta lógica na lista da Fronteira-J, pois sua saída está definida com o valor lógico “1”, contudo, suas entradas estão indefinidas com o valor X.

Para realizar as principais etapas como ativação de falhas, propagação, implicação e justificação de sinais, são utilizados conceitos conhecidos como Cobertura Singular (SC ou *Singular Cover* do inglês), Propagação de Cubo-D (PDC ou *Propagation D-Cube* do inglês) e Cubo-D Primitivo para uma falha (PDCF ou *Primitive D-Cube for a fault* do

inglês). Tais conceitos determinam a interação do conjunto da Notação D para cada porta lógica.



(a) Exemplo de Fronteira-D



(b) Exemplo de Fronteira-J

Figura 12 – Exemplos dos Conceitos de Fronteiras aplicada ao Algoritmo-D

O primeiro conceito Cobertura Singular é definido como o menor número de atribuições às entradas de uma porta lógica para a saída ser “0” ou “1” lógico. O exemplo mostrado na Tabela 2 define o conjunto SC para as operações lógicas AND e OR de duas entradas. Levando em consideração a operação AND, é possível notar que o menor número de atribuições das entradas para se obter “1” na saída é tendo os dois valores da entrada em “1”, e para se obter “0” na saída basta ter “0” lógico em uma das entradas. Analogamente, é possível obter o conjunto para a porta OR e para qualquer outra porta lógica.

Tabela 2 – Exemplo de Cobertura Singular para portas AND e OR de duas entradas.

AND			OR		
A	B	S	A	B	S
1	1	1	0	0	0
0	X	0	1	X	1
X	0	0	X	1	1

O próximo conceito definido como Propagação de Cubo-D é definido como o menor

número de atribuições às entradas de uma porta lógica para propagar um efeito de falha D ou $\overline{D}$ das entradas para as saídas. O exemplo da Tabela 3 define o conjunto PDC para uma porta lógica NOR de duas entradas, em que para propagar um efeito D é preciso aplicar na outra entrada o valor lógico "0" ou outro D para obtenção do seu complemento $\overline{D}$, para propagação de um valor $\overline{D}$ é de forma análoga. Esse conjunto pode ser obtido por meio do conjunto de entradas de uma tabela verdade que causam discrepância entre valor obtido e valor esperado na presença de falhas.

Tabela 3 – Exemplo de Propagação de Cubo-D para porta NOR de duas entradas.

NOR		
A	B	S
0	D	$\overline{D}$
D	0	$\overline{D}$
D	D	$\overline{D}$
0	$\overline{D}$	D
$\overline{D}$	0	D
$\overline{D}$	$\overline{D}$	D

O último conceito conhecido por Cubo-D Primitivo para uma falha é definido como o menor número de atribuições às entradas de uma porta lógica para ativar um efeito de falha D ou $\overline{D}$ presente na sua saída. O exemplo da Tabela 4 define o conjunto PDCF para ativação de falhas de efeito D e $\overline{D}$ em uma porta NOR de duas entradas. Nesse caso, é aplicado o conceito de ativação de falhas, previamente já definido, no qual para ativar uma falha D na saída de uma porta NOR, é preciso todas as entradas em "0" lógico e para ativar uma falha $\overline{D}$ é necessário apenas uma das entradas em "1" lógico.

Tabela 4 – Exemplo de Cubo-D Primitivo para uma falha para porta NOR de duas entradas.

NOR		
A	B	S
0	0	D
1	X	$\overline{D}$
X	1	$\overline{D}$

As principais etapas do Algoritmo-D compreendem as operações básicas de um método ATPG, propagação e justificação, e mais três principais operações, são elas: implicação, intersecção e *backtrack*. As operações base do Algoritmo-D – propagação e justificação, já explanadas – utilizam-se dos conceitos de PDC e SC, respectivamente, para serem efetuadas.

A etapa de implicação é efetuada sempre que uma decisão é realizada e quando uma combinação única de entradas já conhecida de uma porta lógica define o valor lógico

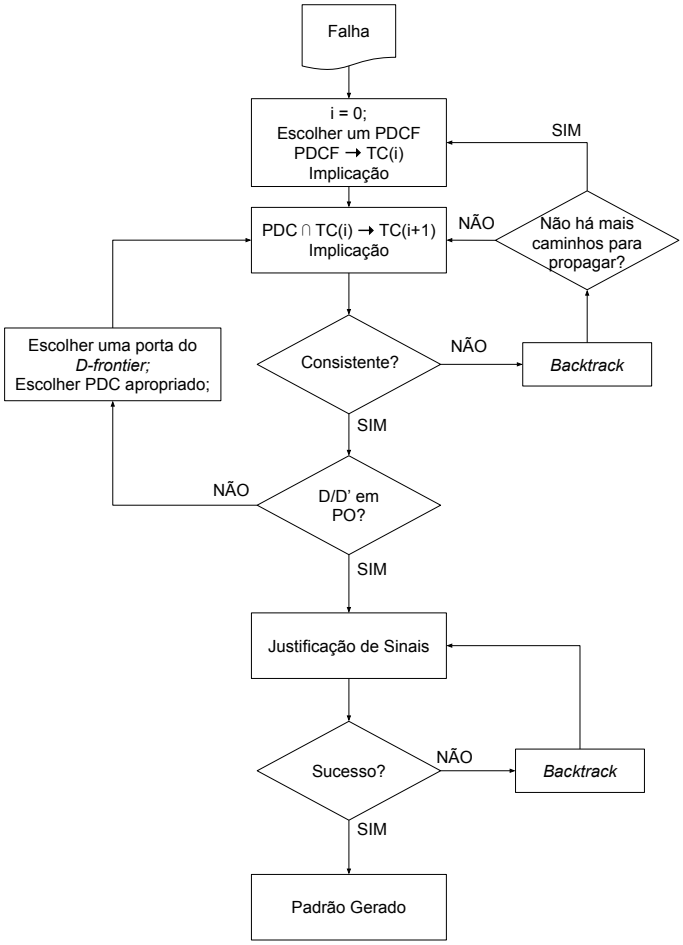
da saída ou, ainda, conhecendo o valor lógico de saída é possível determinar um padrão de entrada único. Essa etapa é realizada para definir valores de sinais unicamente determináveis; dessa forma a etapa de implicação não gera uma escolha, diferente das etapas de propagação onde é escolhido um caminho sensibilizável e justificação onde são escolhidas combinações de sinais internos para se obter o padrão de entrada.

A cada escolha realizada um novo Cubo de teste é gerado, definido como $TC(n)$, onde n é a etapa do processo. Um Cubo de teste é o conjunto de valores Booleanos especificados para testar a falha em questão. No Algoritmo-D, o Cubo de teste é composto pelas entradas primárias e todos os sinais internos do circuito.

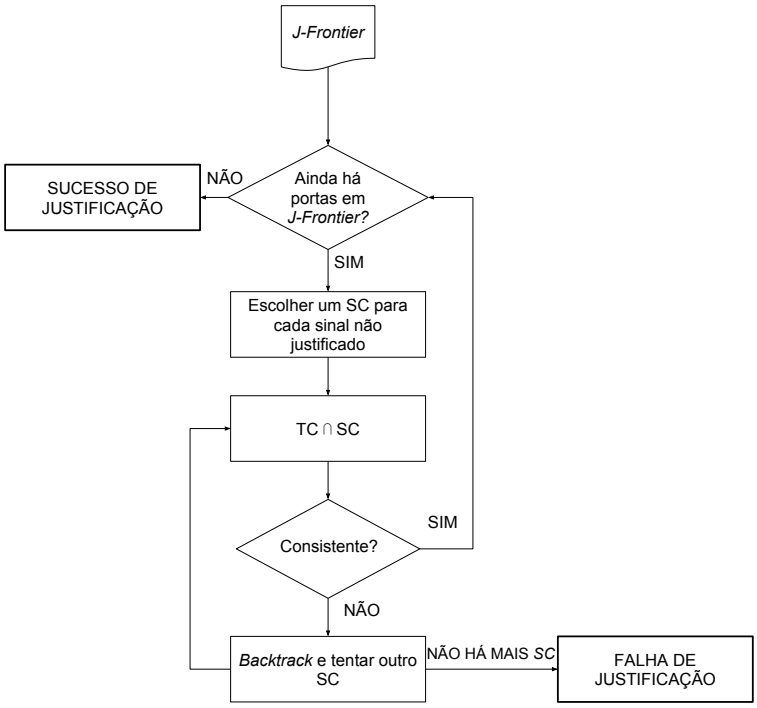
Ao gerar um novo Cubo de teste é realizada a etapa de intersecção, a qual é efetuada a partir da realização de alguma etapa como a propagação, onde será necessário fazer a interseção de um PDC com o $TC(n)$ para geração de um $TC(n+1)$. Porém, etapas que geram escolhas como propagação e justificação podem encontrar estados de inconsistência, nos quais não é possível realizar uma operação de intersecção, ou seja, um certo valor lógico é necessário para justificação de um sinal, no entanto, seu complemento já foi atribuído ao sinal. Quando há uma inconsistência devido a escolhas realizadas, geradas pelas escolhas de PDCF, PDC ou SC, é preciso fazer a etapa de *backtrack*.

A etapa de *backtrack* consiste em refazer a última decisão, até corrigir e voltar às operações do algoritmo, obtendo o padrão de teste ao final ou até atingir um limite de número de *backtracks*. Para evitar um alto gasto de tempo de execução, é definido um limite de *backtracks* a ser realizado por cada análise de falha.

O fluxograma da Figura 13a define a ordem dos processos descritos acima para execução do Algoritmo-D. Dada uma falha, é escolhido um PDCF para criar o primeiro Cubo de teste $TC(0)$ sendo feita uma etapa de implicação após. Depois, é escolhida alguma porta lógica no conjunto Fronteira-D para propagação, escolhendo um PDC devido, fazendo a intersecção com o TC atual para criar um novo Cubo de teste. Se ocorrer uma inconsistência, é realizado um *backtrack* e refeita a escolha tomada, caso contrário, é verificado se o efeito de falha D ou $\overline{D}$ pode ser observado em alguma saída primária. Em caso negativo, é preciso propagar o efeito de falha até chegar em uma saída, escolhendo uma próxima porta da Fronteira-D e um PDC para a propagação da falha. O processo citado é repetido até o efeito da falha atingir uma saída ou até realizar um número máximo de *backtrack*.



(a) Fluxograma do Método Algoritmo-D.



(b) Fluxograma do Método de Justificação de sinais.

Figura 13 – Fluxogramas do processo completo do Algoritmo-D. Adaptado de (ROTH, 1966).

Ao atingir uma saída, o processo de justificação é iniciado. Se o processo falhar, é realizada uma etapa de *backtrack*, em caso positivo é obtido o padrão de teste para a falha analisada. O processo de justificação é definido pelo fluxograma da Figura 13b. A justificação de sinais recebe a lista Fronteira-J, onde terá que justificar os sinais das portas lógicas até obter o padrão de teste. Cada porta lógica contida na lista é analisada. É escolhido um SC para justificar os sinais da porta corrente e realizada uma etapa de intersecção do Cubo de teste com o SC escolhido. Em caso positivo, o processo reinicia até justificar todos os sinais da lista para obtenção do padrão de teste; em caso negativo, ou seja, em caso de inconsistência, é feito um *backtrack* escolhendo outro possível SC. Caso não seja mais possível escolher um SC, o processo de justificação falha.

2.4 Métodos de Satisfatibilidade Booleana aplicada em ATPG

Devido à metodologia de análise topológica das soluções estruturais, esses algoritmos de ATPG utilizados em ferramentas comerciais começaram a apresentar problemas de escalabilidade com o aumento da complexidade dos circuitos. O que ocorre é que, mesmo utilizando apenas falhas do tipo *stuck-at*, que são o tipo de falha padrão para aplicações de geração automática de padrões de teste, a geração de padrões de teste utilizando métodos estruturais pode levar várias semanas para garantir uma boa cobertura de falhas mesmo utilizando um conjunto de servidores (*server-farm* do inglês) para dividir o processamento (BURCHARD et al., 2017). Foi no referido contexto que, buscando uma redução significativa para a execução de ATPGs, algoritmos baseados em SAT começaram a se tornar populares.

SAT ou Método de Satisfatibilidade Booleana (*Boolean Satisfiability Method* do inglês) é um tipo de aplicação que é capaz de inferir se uma expressão Booleana é possível de ser satisfeita ou não. Foi em (LARRABEE, 1992) que esse método foi introduzido já como um novo algoritmo para geração de padrões de teste utilizando falhas do tipo *stuck-at* em circuitos combinacionais.

Enquanto métodos estruturais buscam atribuir valores a sinais e fazem uma busca topológica por intermédio de uma análise de estruturas de dados, métodos como o SAT são classificados como algébricos. Métodos algébricos trabalham com análise de funções Booleanas que representam o circuito, trabalhando com a aplicação e simplificação de álgebra Booleana. O método algébrico clássico é conhecido como método de diferença Booleana (*Boolean difference method* do inglês), que pode ser representado para uma função F qualquer, pela equação a seguir.

$$F(x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_n) \oplus F(x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_n) \tag{2.6}$$

A Equação 2.6 verifica se a entrada x_i da função F gera um erro na saída. Para

isso, são utilizadas as regras básicas de simplificação Booleana. Porém, a natureza da manipulação algébrica da equação torna o método de diferença Booleana inviável para uma ferramenta de geração de teste (LARRABEE, 1992). Para executar uma instância do método SAT, é necessário gerar a expressão Booleana específica do circuito. O método utiliza expressões no formato CNF (Forma Conjuntiva Normal ou *Conjunctive Normal Form* do inglês), pois esse tipo de fórmula é facilmente manipulável via *software*. Fórmulas CNF são expressões compostas exclusivamente de AND, OR e NOT, no formato POS (Produto de Somas ou *Product of Sums*).

Ao aplicar a fórmula CNF, a ferramenta SAT busca determinar se existem valores Booleanos que tornam a expressão verdadeira. Levando em consideração que um resolvedor SAT é uma caixa preta; que, dada uma expressão CNF, ele responde se é possível satisfazer ou não, torna-se crucial para resultado do método ATPG a estruturação da fórmula CNF, já que ela que irá determinar o comportamento do método ATPG. Para representação da estrutura do circuito por meio de uma expressão Booleana CNF, é utilizada o que se conhece por Transformação de Tseitin (TSEITIN, 1983). O objetivo, então, é determinar a estrutura básica de teste, em que as saídas de um circuito livre de falhas são comparadas com o circuito na presença de falhas, por intermédio de uma disjunção exclusiva.

Através da Transformação de Tseitin é possível definir a expressão CNF para cada porta combinacional, levando em consideração a sua estrutura: entradas e saídas. Como exemplo, é descrito como é feita a conversão para o formato CNF para uma porta lógica AND de duas entradas, iniciando com sua expressão Booleana básica.

$$Z = X \cdot Y \; ou \; Z \leftrightarrow X \cdot Y \tag{2.7}$$

Baseado na igualdade $P = Q$, onde P e Q são proposições lógicas genéricas, é possível reescrever como $(P \rightarrow Q) \cdot (Q \rightarrow P)$ com base na regra de equivalência lógica. Aplicando a regra de equivalência lógica na Equação 2.7, pode-se reescrever a equação para porta lógica AND da seguinte forma:

$$Z = X \cdot Y \leftrightarrow (Z \rightarrow (X \cdot Y)) \cdot ((X \cdot Y) \rightarrow Z) \tag{2.8}$$

Depois dessa etapa, é preciso converter todas as operações de implicação para operações de disjunção (OU lógico). Para isso, é aplicada a equivalência lógica que diz que a implicação $P \rightarrow Q$ é equivalente à expressão $\overline{P} + Q$. Assim, pode-se concluir a fórmula CNF, reescrevendo a Equação 2.8 como:

$$(Z \rightarrow (X \cdot Y)) \cdot ((X \cdot Y) \rightarrow Z) \leftrightarrow (\overline{Z} + X) \cdot (\overline{Z} + Y) \cdot (\overline{X} + \overline{Y} + Z) = 1 \tag{2.9}$$

A Equação 2.9 resulta no valor lógico "1", se os valores lógicos das variáveis X, Y e Z estiverem consistentes com a tabela verdade da porta lógica AND (LARRABEE, 1992). Expressões CNF são divididas em cláusulas e essas são partículas de somas da fórmula. Cláusulas são classificadas de acordo com o número de elementos. Por exemplo, cláusulas com um, dois ou três elementos são chamadas de unárias, binárias ou ternárias, respectivamente. No exemplo anterior, representado pela fórmula CNF da porta lógica AND, a Equação 2.9 possui três cláusulas, sendo duas cláusulas binárias e uma cláusula ternária.

Utilizando a mesma lógica é possível obter a expressão CNF para qualquer função lógica. A partir de fórmulas CNF de portas lógicas básicas de duas entradas, é possível obter fórmulas para as mesmas portas de n entradas, apenas fazendo pequenas alterações. Utilizando-se do mesmo exemplo descrito anteriormente: para uma fórmula CNF de uma porta lógica AND de três entradas X, Y e W e uma saída Z, é possível reescrever a Equação 2.9 como:

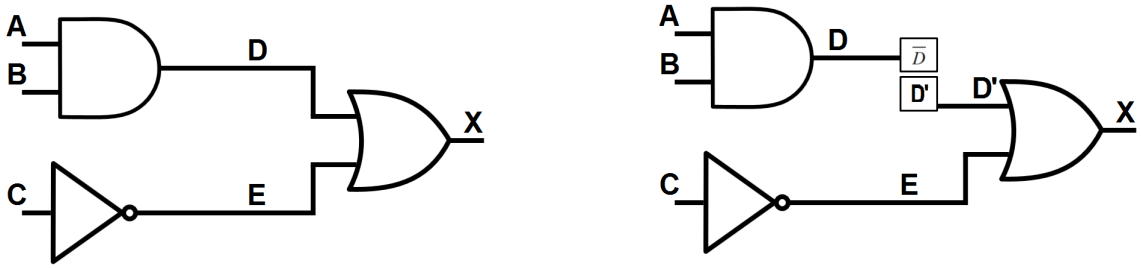
$$(\overline{Z} + X) \cdot (\overline{Z} + Y) \cdot (\overline{Z} + W) \cdot (\overline{X} + \overline{Y} + \overline{W} + Z) = 1 \tag{2.10}$$

Na Equação 2.10 foi necessário adicionar uma variável W. Assim, é acrescentada uma cláusula binária e aumentando a cláusula ternária para quaternária, enquanto se mantém a lógica estrutural da fórmula CNF original para AND de duas entradas. Com a fórmula CNF para cada porta lógica já definida, é preciso definir a fórmula para o circuito. Para obter a fórmula CNF para um circuito livre de falhas, é feita uma análise das saídas primárias do circuito percorrendo todos os nodos do circuito fazendo uma conjunção de todas fórmulas CNF.

A Figura 14a representa um circuito livre de falhas. Como exemplo, para obter a fórmula CNF, a análise é feita a partir da saída primária X percorrendo todas as demais portas. Então, a fórmula CNF para a saída do circuito é a conjunção das fórmulas das portas lógicas OR, AND e NOT. Podendo ser escrita como:

$$(X + \overline{D}) \cdot (X + \overline{E}) \cdot (\overline{X} + D + E) \cdot (\overline{D} + A) \cdot (\overline{D} + B) \cdot (D + \overline{A} + \overline{B}) \cdot (C + E) \cdot (\overline{C} + \overline{E}) \tag{2.11}$$

Ao conseguir representar a estrutura do circuito livre de falhas, vide Figura 14b, é preciso, então, representar a porção do circuito afetado pela falha para gerar o circuito para o teste. A porção afetada pela falha é chamada de cone da falha. Para representá-lo dentro do circuito (Figura 15), um conjunto de cláusulas são definidas para este e para os demais modelos de ATPG baseados em SAT (CHEN; MARQUES-SILVA, 2013).



(a) Circuito Combinacional *Fault free*

(b) Circuito Combinacional com a presença de *Stuck-at-1*

Figura 14 – Exemplo de Circuito Combinacional para Geração de Fórmula CNF

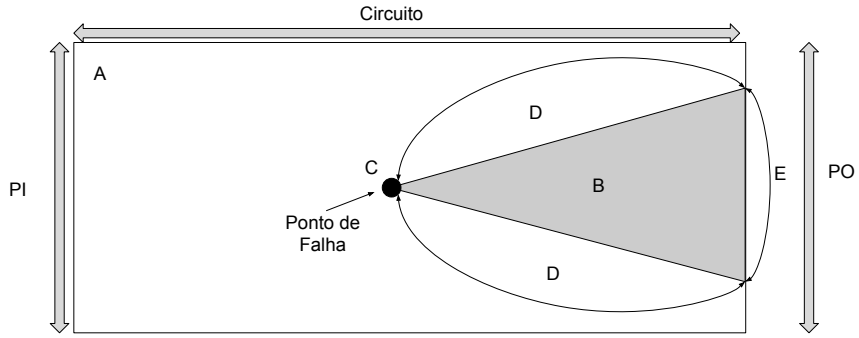


Figura 15 – Representação do cone da falha em um circuito. Adaptado de (CHEN; MARQUES-SILVA, 2013)

A: o conjunto de fórmulas CNF para a função livre de falhas do círcuito, entre entradas e saídas primárias.

B: o conjunto de fórmulas CNF que representam condições de propagação para a falha no ponto C para pelo menos uma das saídas primárias.

C: fórmula CNF que representa a ativação da falha.

D: o conjunto de fórmulas CNF que não estão presentes no cone da falha, porém podem influenciar na propagação do efeito da falha.

E: o conjunto de fórmulas CNF que representam as saídas primárias, onde o estado errôneo produzido pela falha pode ser propagado para pelo menos uma delas.

De acordo com as cláusulas definidas acima, é possível obter a fórmula CNF que representa o cone o qual a falha pode afetar. Como visto na Figura 14b é feita uma quebra do sinal, onde até aquele ponto o valor original é considerado, a partir do ponto da falha é considerado então o valor de efeito da falha. O nodo sem falha D passa agora a ter falha, representado pelo rótulo D'. O novo rótulo D' é representado com uma cláusula unária

para indicar o comportamento da falha em questão (LARRABEE, 1992). A expressão pode ser definida como:

$$(X' + \overline{D'}) \cdot (X' + \overline{E}) \cdot (\overline{X'} + D' + E) \cdot (D') \cdot (C + E) \cdot (\overline{C} + \overline{E}) \tag{2.12}$$

Para testar a dada falha *stuck-at-1* representada na Figura 14b, é preciso encontrar um padrão de entrada para o qual as saídas do circuito livre de falhas e do falho sejam diferentes. É preciso estabelecer uma fórmula que garanta a comparação dos dois tipos do circuito e que seja fácil a detecção dessa diferença de valores lógicos. É realizada uma operação de disjunção exclusiva para verificar tal diferença. Quando a saída primária do circuito apresentar o valor lógico "0", a falha foi mascarada; e quando apresentar o valor lógico "1", o erro foi verificado e o padrão gerado testa a falha em questão. A Equação 2.13 representa na forma CNF essa operação XOR entre circuitos sem e com falha. É criada uma variável BD que representa a saída da operação XOR entre as saídas X e X'. Se for possível satisfazer a Equação 2.13, ou seja, há uma combinação de valores lógicos em que o resultado seja "1", então há pelo menos um padrão de entrada para o circuito que verifique o erro decorrente a falha na saída.

$$\begin{aligned}
&(X' + \overline{D'}) \cdot (X' + \overline{E}) \cdot (\overline{X'} + D' + E) \\
&\cdot (D') \cdot (C + E) \cdot (\overline{C} + \overline{E}) \\
&\cdot (X + \overline{D}) \cdot (X + \overline{E}) \cdot (\overline{X} + D + E) \\
&\cdot (\overline{D} + A) \cdot (\overline{D} + B) \cdot (D + \overline{A} + \overline{B}) \\
&\cdot (\overline{X} + X' + BD) \cdot (X + \overline{X'} + BD) \\
&\cdot (\overline{X} + \overline{X'} + \overline{BD}) \cdot (X + X' + \overline{BD})
\end{aligned} \tag{2.13}$$

O formato CNF exigido pela aplicação SAT possui variáveis extras. Tais redundâncias aplicam restrições adicionais para a validação da fórmula, o que auxilia no processo de satisfazer a fórmula a encontrar um padrão de teste (LARRABEE, 1992). Com essa implementação clássica de teste para ATPGs baseados em SAT já é possível a obtenção de padrões de teste. Modificações de expressões CNF podem ser adotadas adicionando condições que guiem o resolvedor SAT a certas resoluções, como, por exemplo, a obtenção de um conjunto de padrões menores. Abordagens que visam à otimização de fórmulas CNF, dentre outras aplicações atuais, serão abordadas na próxima seção de estado-da-arte.

3 Estado da Arte

Quando o problema via CNF-SAT para geração de padrões de teste foi definido em (LARRABEE, 1992), o método SAT não era maduro o suficiente para competir com as ferramentas estruturais da época. Porém, ao longo dos anos, o desenvolvimento de resolvedores SAT cada vez mais eficientes como Chaff (MOSKEWICZ et al., 2001), GRASP (MARQUES-SILVA; SAKALLAH, 1999) e MINISAT (EéN; SöRENSSON, 2004) tornaram ATPGs baseados em SAT mais competitivos e mais práticos. Dessa forma, resolvedores SAT atuais tornaram-se eficientes ao ponto de que exista pouco espaço para acelerar seu tempo de execução para satisfazer uma fórmula, de acordo com projetos atuais (MATSUNAGA, 2017). Apesar de obter um tempo de execução menor em relação aos métodos estruturais, ATPGs baseados em SAT possuem dificuldade para geração de um conjunto reduzido de padrões de teste. Com isso, as soluções mais atuais de ATPGs baseadas em SAT buscam aprimorar as fórmulas CNF ou como estas fórmulas serão entregues para a instância SAT, buscando cada vez mais a ampliação de performance de ATPGs e o contorno de desvantagens. A seguir, serão abordados exemplos de soluções atuais que comprovam tal constatação.

O primeiro exemplo de aplicação é (BURCHARD et al., 2017), que tem por objetivo avaliar a eficiência de *D-chains* em ATPGs baseados em SAT. *D-chain* nada mais é que a representação da propagação da falha até uma saída primária. O modo como é computado o *D-chain* reflete-se na expressão CNF e, consequentemente, na eficiência do processo de satisfatibilidade booleana.

D-chains adicionam informações redundantes baseadas em informações estruturais para auxiliar o processo de resolução SAT (LARRABEE, 1992). Existem diferentes maneiras de computar um *D-chain*, como, por exemplo: Implicação Progressiva (*Forward Implication*), Implicação Reversa (*Backward Implication*) e uma forma híbrida combinando as duas maneiras. Basicamente, a implicação progressiva busca propagar o efeito de falha em direção a uma saída primária. De forma complementar, a implicação reversa busca ir de uma saída primária até o *fault site*. Por fim, a forma híbrida combina essas duas técnicas, o que faz com que exista uma pequena redução no número de cláusulas da fórmula.

Outro conceito utilizado por (BURCHARD et al., 2017), *D-chain* indireto, busca reduzir informações redundantes retirando totalmente a parte que representa o circuito falho. Com isso, é apenas gerada a fórmula para o circuito livre de falhas e as diferenças causadas pela falha. É provado que, para a metodologia utilizada, houve um ganho em tempo de resolução de 25% a 35% para falhas *stuck-at*.

Uma outra alternativa foi proposta em (CHEN; MARQUES-SILVA, 2013), a qual propõe uma abordagem híbrida para o ATPG, o TG-System. Métodos estruturais possuem um ótimo desempenho para a detecção de falhas fáceis (*easy-faults*), ao passo que métodos baseados em SAT têm um desempenho melhor na detecção de falhas difíceis (*hard-faults*) (DRECHSLER et al., 2008). Com esse intuito, foi proposto um ATPG híbrido, onde as falhas são analisadas utilizando métodos de análise de testabilidade, em que falhas fáceis são enviadas para o módulo de ATPG estrutural, enquanto as demais vão para o baseado em SAT. O ATPG estrutural utilizado para a aplicação escolhido foi o ATALANTA (LEE; HA, 1993), um ATPG baseado no algortimo FAN (FUJIWARA; SHIMONO, 1983), enquanto para o módulo SAT foi utilizado o TG-PRO (CHEN; MARQUES-SILVA, 2009). Por fim, foi comparado com um método estrutural atual, um método que visa a paralelização via *threads* e um baseado em SAT. Para os *benchmarks* ISCAS85 (BRYAN, 1985) e ISCAS89 (BRGLEZ; BRYAN; KOZMINSKI, 1989) o método em questão obteve um tempo de execução consideravelmente menor que as demais aplicações, enquanto no *benchmark* ITC99 (CORNO; REORDA; SQUILLERO, 2000), na maioria dos casos, obteve um desempenho semelhante ou superior aos demais. Em apenas dois casos, no circuito b17, ficou atrás do método baseado em SAT e, no circuito b18, o qual apresenta o maior número de falhas entre eles, ficou atrás apenas do método paralelizado via *threads*.

Outra abordagem (MATSUNAGA, 2017), propõe uma nova forma para geração de fórmulas CNF, visando uma redução do tempo de execução da instância SAT. Foi constatado pelo autor que, muitas vezes, o tempo para a geração da fórmula CNF acaba sendo maior que a execução da instância SAT. É proposto o algoritmo de compartilhamento CNF (*CNF-sharing algorithm*), que busca reutilizar a mesma fórmula CNF para mais de uma falha. É visto que as condições para falhas propagarem para a saída podem ser as mesmas, mas como a falha vai ser ativada pode ser diferente. É definida uma heurística onde falhas que compartilham fórmulas CNF estão contidas em um mesmo MFFC (*maximal fan-out free cone*), ou seja, o cone máximo de uma região livre de *fan-out*.

Todas as falhas dentro de um MFFC passam pelas mesmas raízes e as condições de propagação são as mesmas. Por fim, a ferramenta baseada em SAT utilizando o *CNF-sharing algorithm* foi comparada com a ferramenta anterior, TG-System, como também as demais já comparadas no trabalho anterior (CHEN; MARQUES-SILVA, 2013). A ferramenta de (MATSUNAGA, 2017) obteve os melhores resultados, apresentando em média uma redução temporal de 2.5 vezes, principalmente nos *benchmarks* ITC99, onde o TG-System obtinha alguns resultados não satisfatórios. Esse comportamento deve-se ao fato de que o método proposto é mais eficiente em circuitos com um número maior de falhas, pois são os que apresentam uma maior probabilidade de compartilhamento de fórmulas CNF.

Em (ALI; HUSSEIN; ALI, 2016) foi proposto a paralelização do processo de ATPG

utilizando GPU (*Graphics Processing Unit*, ou Unidade de Processamento Gráfico). A paralelização se dá nas fórmulas CNF, baseando-se no conceito em que quanto menor a fórmula, mais rápida a instância SAT será resolvida. Foram implementados dois métodos. O primeiro busca paralelizar as cláusulas, enquanto o segundo método busca paralelizar tanto as cláusulas quanto os seus literais. Os seus resultados dependem bastante do desempenho da GPU utilizada, porém já são melhores que aplicações mais tradicionais baseadas em SAT, como as anteriores.

Outra abordagem que visa a uma maior exploração de fórmulas CNF foi proposta em (EGGERSGLÜSS et al., 2016). O autor propõe uma otimização para ATPG baseada em SAT de múltiplas falhas simultâneas que têm por objetivo a obtenção de um conjunto compacto de padrões de teste com uma alta cobertura de falhas por meio de métodos de compactação de teste. Essa nova formulação de múltiplas falhas aplicadas em um ATPG baseado em SAT consegue contornar a desvantagem que aplicações SAT têm para geração de um conjunto de padrões reduzidos.

É demonstrado que ATPGs estruturais estado da arte além de terem problemas para gerar padrões para falhas difíceis, também sofrem para obter uma alta cobertura de falhas. O método proposto demonstra que é possível obter além de uma cobertura de falhas alta, um conjunto reduzido de padrões de teste. Pelos experimentos realizados, é demonstrado que – até para um circuito industrial – a técnica proposta pode alcançar até 26% de redução no tamanho do conjunto de teste em comparação a uma ferramenta ATPG industrial.

4 Métodos ATPG Implementados

Neste capítulo, os principais módulos da ferramenta ATPG desenvolvida serão descritos. Para alcançar o objetivo proposto – analisar a eficiência de métodos híbridos de geração de padrões de teste – cinco métodos ATPG foram desenvolvidos. A Seção 4.1 aborda os módulos individualmente, descrevendo dados de entrada, fluxograma de funcionamento e saída de dados. Enquanto na Seção 4.2 é abordada a organização, bem como o funcionamento dos métodos híbridos que serão utilizados para análise neste trabalho.

4.1 ATPGs Implementados

Nesta seção serão abordados os métodos ATPGs implementados. Aspectos como funcionamento de cada módulo, entrada e saída e parâmetros configuráveis serão abordados.

4.1.1 Estrutura Geral das Aplicações ATPG

Neste trabalho, todos ATPGs implementados compartilham de algumas estruturas de dados e métodos. A estrutura básica genérica para os ATPGs implementados pode ser vista na Figura 17. A estrutura geral pode ser dividida em: pré-processamento, sintetização de padrões de teste e pós-processamento.

A parte de pré-processamento consiste em outras três partes: recebimento de arquivos e parâmetros de entrada; mapeamento do circuito em uma estrutura de dados e a obtenção do conjunto de falhas a serem analisadas. Na primeira etapa, é fornecida para a ferramenta a descrição da estrutura lógica do circuito por intermédio de um arquivo do tipo *bench* ou *verilog*, um parâmetro configurável que depende do módulo ATPG utilizado e um arquivo de biblioteca de portas lógicas utilizadas para mapeamento de funções lógicas do circuito no formato *genlib* (vide Figura 16 para exemplo).

Com os referidos dados, é criado um grafo bidirecional representando a estrutura lógica do circuito, em que seus nodos são as portas lógicas mapeadas utilizando a biblioteca informada; e as arestas, os sinais entre as portas. Com isso, a etapa de geração do conjunto de falhas é inicializada, onde é executado o método de *Fault Collapsing*. O processo de *Fault Collapsing* implementado utiliza-se de um arquivo de entrada que possui padrões de equivalência e dominância entre falhas, a fim de reduzir o conjunto final de falhas a serem processadas. Com o conjunto de falhas a ser analisado e o parâmetro configurável definido, o módulo ATPG inicia o processo de análise e síntese de padrões de teste, em

que ao final irá produzir um conjunto de padrões de teste correlacionados com falhas analisadas e um conjunto de falhas abortadas.

```
# 180 nm Generic Library
# Download from http://crete.cadence.com
# Copyright 2003, Cadence Design Systems - All Rights Reserved
# (Single-output logic gates converted to GENLIB by Alan Mishchenko.)
GATE INV    1  Y=!A;                        PIN * INV 1 999 1 0 1 0
GATE NOR2   1  Y=!(A+B);                    PIN * INV 1 999 1 0 1 0
GATE NOR3   1  Y=!(A+B+C);                  PIN * INV 1 999 1 0 1 0
GATE NOR4   1  Y=!(A+B+C+D);                PIN * INV 1 999 1 0 1 0
GATE NAND2  1  Y=!(A*B);                    PIN * INV 1 999 1 0 1 0
GATE NAND3  1  Y=!(A*B*C);                  PIN * INV 1 999 1 0 1 0
GATE NAND4  1  Y=!(A*B*C*D);                PIN * INV 1 999 1 0 1 0
GATE OAI21  1  Y=!((A0+A1)*B0);             PIN * INV 1 999 1 0 1 0
GATE OAI22  1  Y=!((A0+A1)*(B0+B1));        PIN * INV 1 999 1 0 1 0
GATE OAI33  1  Y=!((A0+A1+A2)*(B0+B1+B2));  PIN * INV 1 999 1 0 1 0
GATE AOI21  1  Y=!(A0*A1+B0);               PIN * INV 1 999 1 0 1 0
GATE AOI22  1  Y=!(A0*A1+B0*B1);            PIN * INV 1 999 1 0 1 0
GATE AOI33  1  Y=!((A0*A1*A2)+(B0*B1*B2));  PIN * INV 1 999 1 0 1 0
GATE MX2    1  Y=(A*B)+(S0*B)+(!S0*A);      PIN * UNKNOWN 1 999 1 0 1 0
GATE XOR2   1  Y=(A*!B)+(!A*B);             PIN * UNKNOWN 1 999 1 0 1 0
```

Figura 16 – Exemplo de biblioteca de portas lógicas no formato *genlib*.

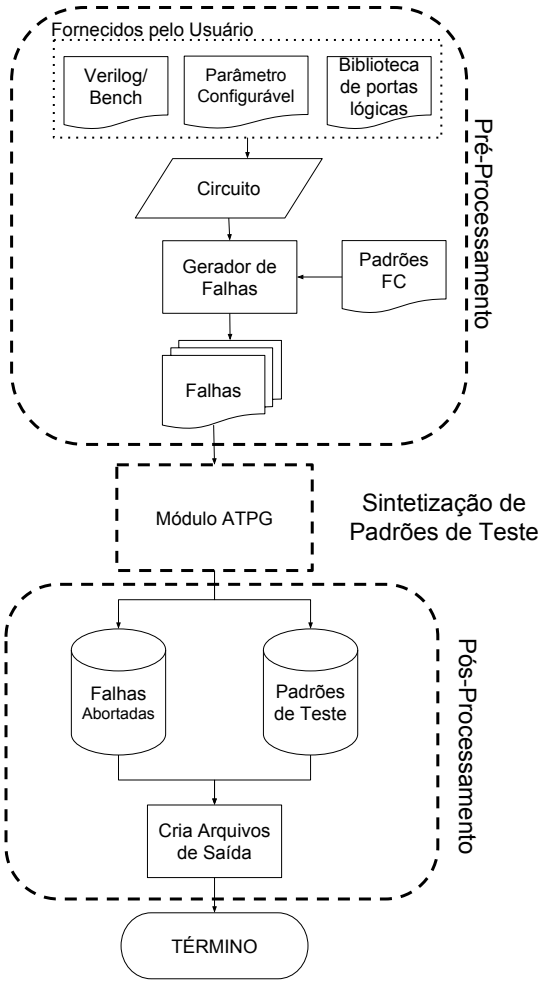


Figura 17 – Fluxograma Genérico para Estrutura de ATPGs implementados.

Ao término do processamento das falhas pelo módulo ATPG, é realizada a etapa de pós-processamento, que consiste na obtenção de informações da execução do método ATPG, bem como na compilação dos dados de padrões gerados e falhas abortadas, gerando arquivos de saída. São produzidos dois arquivos de saída: um contendo os padrões obtidos e a ordem de sinais de entrada consideradas pelo método e outro contendo o *log* de execução do método contendo dados relevantes, como cobertura de falhas, quantidade de padrões gerados e tempo de execução.

Exemplo de arquivos de saídas da ferramenta pode ser visto na Figura 18. A Figura 18a representa o arquivo de saída que contém o conjunto de padrões de teste obtidos pela execução da ferramenta e a ordem das entradas primárias consideradas na geração do conjunto. A Figura 18b apresenta o arquivo de saída contendo os dados sobre a execução do método em questão. Os dados obtidos pela execução podem ser divididos em: informação sobre a estrutura do circuito, como número de sinais e portas lógicas, e sobre a execução do ATPG, como tamanho do conjunto de padrões e cobertura de falhas.

```
1 c Vector File c432ResultRandomSATv2_l-t=8-1000.vecout
2 c runtime: 5,2665   seconds
3 c File created at 18/05/2018 - 10:22 -AM
4 c Inputs Order: 1gat 4gat 11gat 17gat 24gat 30gat 37gat 43gat 50gat
  60gat 66gat 73gat 79gat 86gat 92gat 99gat 105gat 112gat 115gat
5
6 1100000000100000101001000100100010101
7 1000000000101000100000000000000110000
8 1111111010110000111100110101100010111
9 1100001110110000101100000000110010111
10 1100000011110000101100000100100010111
11 1111100000011110011111101010100110010
12 1111100000011111100111100000011111100
13 1010101010101010111100111101110011101
14 1100001111111111011000001010110010111
15 1101011101110111010011111011001100110
```

(a) Arquivo de saída com padrões de teste gerados.

```
 1 ************************************************************
 2 *                                                          *
 3 *                  ATPG Random w/ SatTool v2               *
 4 *     Grupo de Sistemas Digitais e Embarcados - GSDE       *
 5 *         Centro de Ciências Computacionais - C3           *
 6 *        Universidade Federal do Rio Grande - FURG         *
 7 *                 Gabriel Soares Porto                     *
 8 *                                                          *
 9 ************************************************************
10
11
12 ***** SUMMARY OF TEST PATTERN GENERATION RESULTS *****
13 1. Circuit Structure:
14 Circuit Name               : c432
15 Number of Gate             : 160
16 Number of Signals          : 432
17 Number of Inputs           : 36
18 Number of Outputs          : 7
19
20
21 2. Test Pattern Generation Results
22 Total Number of Faults     : 864
23 Number of Collapsed Faults : 524
24 Number of Test Patterns    : 80
25 Number of unSAT Faults     : 4
26 Number of Aborted Faults   : 0
27 Fault Coverage             : 99,2366%
28
29
30 3. Memory Used             : 35 MB
31
32
33 4. CPU Time
34 Total                      : 5,2665   seconds
35 ************************************************************
```

(b) Arquivo de saída com dados de execução do módulo ATPG.

Figura 18 – Exemplos de arquivos de saída gerados pela ferramenta ATPG implementada.

4.1.2 Geração Aleatória de Padrões de Teste

Abordagens aleatórias têm o intuito de reduzir a carga para aplicações de geração de teste principais, como metodologias estruturais e baseadas em SAT. Devido à baixa complexidade, seu papel de redução de carga é cumprido com êxito, tornando-se uma ótima alternativa para um processamento inicial das falhas. Para tal propósito, dois módulos de ATPGs aleatórios foram implementados: o primeiro utiliza métodos estruturais e o segundo é baseado em resolvedores SAT.

O módulo Aleatório Estrutural, representado pelo fluxograma da Figura 19, utiliza-se de estruturas de dados como grafo para representação da estrutura do circuito. Para realizar esse procedimento, são utilizados conceitos de propagação de sinais e de efeitos de falha, utilizando a notação D e definições do Algoritmo-D para simular o comportamento do padrão gerado para uma certa falha. Enquanto para o módulo Aleatório baseado em resolvedores SAT, demonstrado pelo fluxograma na Figura 20, utiliza-se de manipulação de fórmulas CNF e execução via resolvedor SAT para validar um dado padrão aleatório para uma certa falha. A seguir, o procedimento utilizado para geração de padrões de teste de forma aleatória será descrito.

O método Aleatório Estrutural – antes de começar – recebe um conjunto de falhas, a estrutura de dados do circuito e o parâmetro configurável do método. O parâmetro configurável para qualquer uma das duas aplicações aleatórias é o número limite de vezes seguidas em que o método pode resultar em tentativas sem sucesso de cobertura. Uma tentativa sem sucesso é definida pelo comportamento quando um padrão aleatório gerado não cobre nenhuma falha do conjunto atual de falhas analisado. Desse modo, a aplicação é terminada quando esse comportamento é repetido de acordo com o número de vezes imposto pelo parâmetro de limite.

Ao se iniciar o processo, é definido um contador que será utilizado para verificar se o limite foi alcançado. Em seguida, é gerado o padrão de teste e simulado seu comportamento no circuito utilizando um arquivo de entrada que define padrões de cobertura singular (SC ou*Singular Cover*), conceito básico do Algoritmo-D descrito na Seção 2.3, que define meios de propagação de sinais. A partir deste momento, todas as falhas do conjunto atual, uma a uma, são inseridas no circuito e é simulado seu comportamento, utilizando arquivos de padrões definidos para conceitos como propagação de cubo-D (PDC ou *Propagation D-Cube*) e cubo-D primitivo para uma falha (PDCF ou *Primitive D-Cube for a fault*) que definem meios de propagação e ativação de falhas.

Toda falha verificada com sucesso é correlacionada ao padrão aleatório atual e marcada para remoção da lista de falha atual. Quando todas as falhas do conjunto forem analisadas e um certo subconjunto do conjunto de falhas em análise for correlacionado ao padrão aleatório atual, diz-se, então, que a análise obteve sucesso.

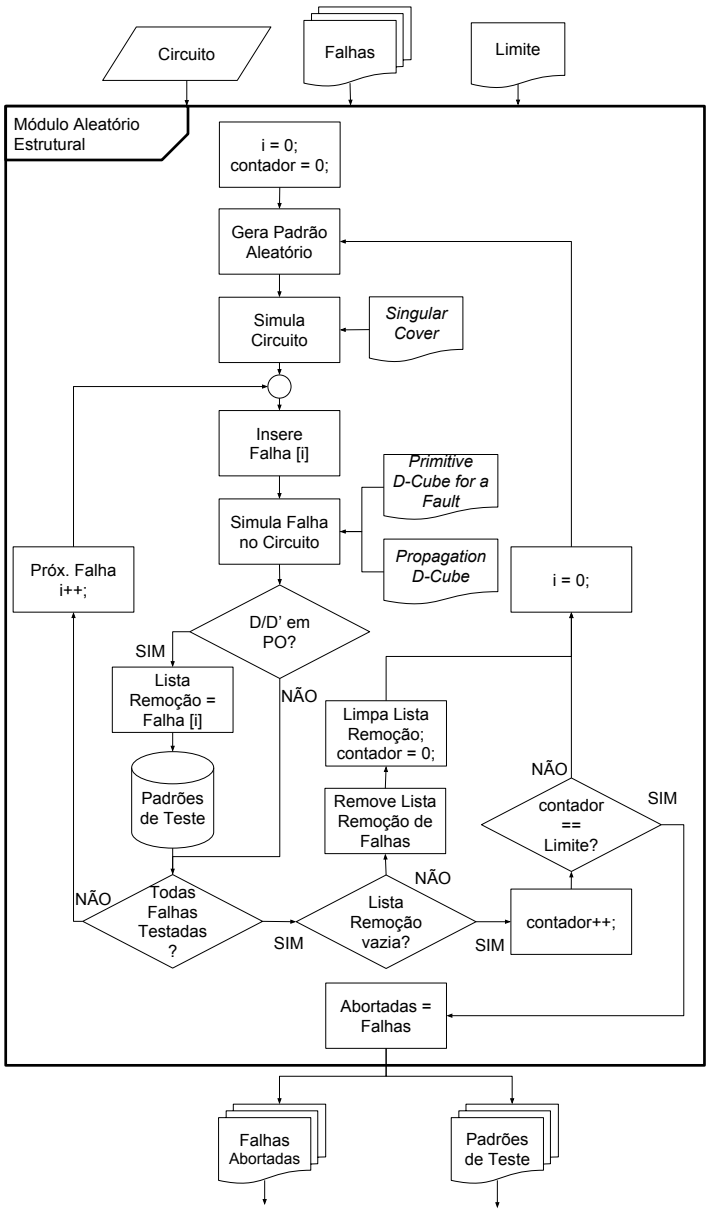


Figura 19 – Fluxograma ATPG Aleatório Estrutural.

Nesse caso, o subconjunto em que foi encontrada uma solução é removido do conjunto a ser analisado e o processo reinicia gerando um próximo padrão aleatório, reiniciando o contador de limite. Caso nenhuma falha seja correlacionada ao padrão aleatório atual, o contador é incrementado e é verificado se o contador coincide com o limite imposto: em caso negativo, o processo reinicia e, em caso positivo, o processo finaliza definindo o conjunto de falhas atual em falhas que foram abortadas pelo processo.

O módulo ATPG Aleatório baseado em SAT, representado na Figura 20, é similar à execução do processo Aleatório Estrutural, já que se trata da mesma metodologia. A diferença entre os métodos está na parte de simulação do padrão gerado e na falha inserida, em que o método anterior era por algoritmos de busca em grafos e neste é utilizado um resolvedor SAT. Para esta aplicação, e posteriores, que irão ser abordadas a seguir de

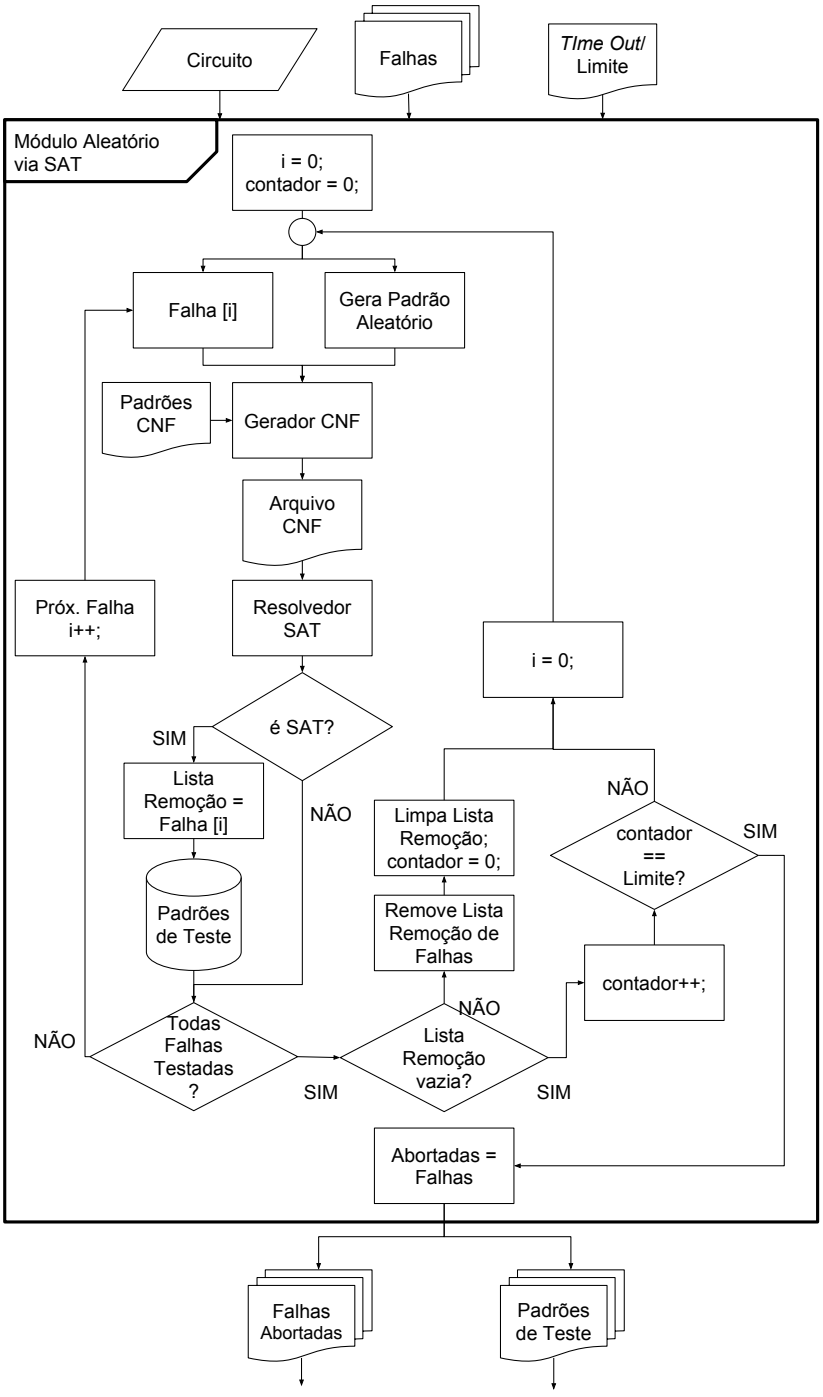


Figura 20 – Fluxograma ATPG Aleatório baseado em SAT.

mesma metodologia de simulação via SAT, é utilizado o SAT4J (Université d'Artois, 2017), uma biblioteca Java que implementa o resolvedor MINISAT (EéN; SöRENSSON, 2004). É necessário nesse caso – como em outros que utilizam SAT – um parâmetro extra chamado de *Time out*, o qual define o tempo que o resolvedor possui para análise de cada expressão CNF atribuída a ele.

Para simulação do circuito é necessário gerar a fórmula CNF para o circuito, a falha e o padrão aleatório gerado, seguindo a metodologia apresentada na Seção 2.4. Para tal função foi implementado um gerador CNF, o qual utiliza um arquivo de entrada onde padrões de fórmulas CNF foram definidas e obtidas por via de metodologias apresentadas neste trabalho.

Como saída, o gerador CNF cria uma arquivo contendo a fórmula que serve de entrada para o resolvedor SAT. Esse, ao analisar a fórmula, resulta em um estado positivo se a falha é correlacionada ao padrão aleatório atual e é marcada para ser excluída do conjunto de falhas a analisar. Quando todas as falhas são analisadas, o método aleatório baseado em SAT segue o mesmo caminho do método aleatório estrutural, ou seja, quando o subconjunto de falhas correlacionadas não é vazio, o processo é reiniciado, caso contrário o contador é incrementado e é verificado se o valor do contador é igual ao limite imposto. Ao alcançar o valor do limite o método é finalizado, atribuindo o conjunto atual de falhas como abortadas.

4.1.3 ATPG Estrutural Clássico: Algoritmo-D

O método estrutural adotado para este trabalho foi o Algoritmo-D (ROTH, 1966). Esse método implementa conceitos básicos e importantes para geração de padrões de teste, a exemplo dos conceitos de propagação de efeito de falhas e justificação de sinais. Para o presente trabalho, o módulo do Algoritmo-D está representado pela Figura 21.

Previamente foi discutido que a parte inicial do processo é comum para qualquer módulo ATPG implementado, como a criação do grafo do circuito e a geração do conjunto de falhas através do método de *Fault Collapsing*. Nesse caso, o que difere é o parâmetro configurável, sendo no Algoritmo-D o limite para *backtracks* a serem executados pelo método. Quanto maior o valor do limite de *backtrack*, maior será o tempo dispendido, visto que o processo de *backtracking* consiste em voltar atrás em escolhas tomadas e buscar novos caminhos.

Essencialmente, uma falha por vez é enviada para o processo do Algoritmo-D, onde irá executar os passos do algoritmo. A implementação feita, para realizar certos passos do algoritmo, utiliza arquivos de entrada contendo padrões para SC, PDC e PDCF. Ao finalizar o processo do Algoritmo-D de acordo com o limite de *backtracks* imposto, o processo resulta em sucesso ou não. Em caso de sucesso, o padrão obtido é correlacionado

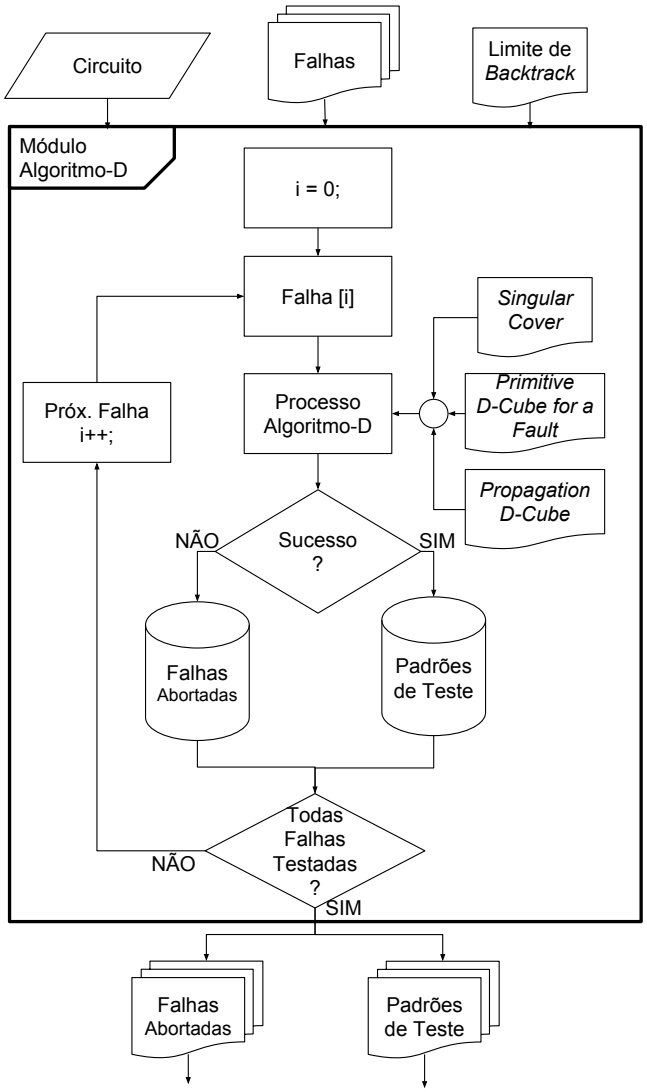


Figura 21 – Fluxograma ATPG Clássico - Algoritmo-D.

à falha analisada; em caso negativo, a falha em análise é contabilizada como falha abortada pelo processo. Ao final, como todo processo, são criados os arquivos de saída contendo os padrões gerados pelo Algoritmo-D e as informações relevantes da execução do método.

4.1.4 ATPG baseado em SAT

Neste trabalho foram implementadas duas soluções em ATPG baseados em SAT, uma referente à implementação clássica de (LARRABEE, 1992) e uma segunda, na qual são explorados meios de compactação de padrões de teste. Os módulos de ATPG baseados em SAT recebem como parâmetro configurável o valor de *Time Out* em milissegundos, que define o tempo máximo dentro do qual o resolvedor SAT tem para analisar cada expressão CNF.

O módulo ATPG baseado em SAT Clássico, representado no fluxograma da Figura 22, inicialmente recebe a estrutura do circuito mapeado e o parâmetro *Time Out*. A partir

disso, uma falha por vez é analisada. O gerador CNF cria a expressão CNF para o circuito em questão inserindo a falha a ser analisada, considerando a metodologia ATPG baseada em SAT apresentada por (LARRABEE, 1992).

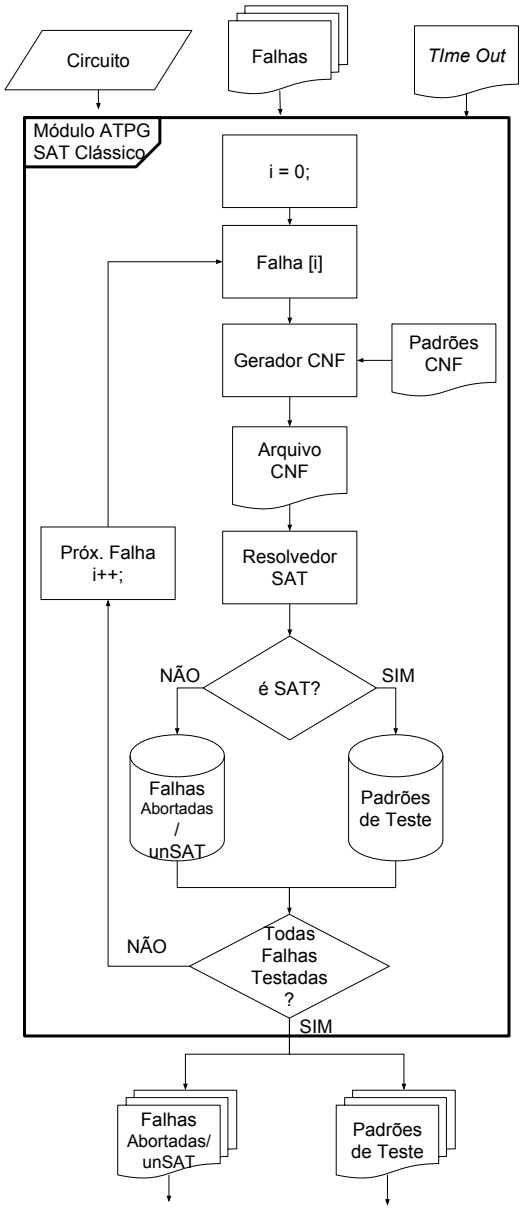


Figura 22 – Fluxograma ATPG baseado em SAT Clássico.

Ao gerar o arquivo contendo a expressão CNF, o resolvedor SAT começa a analisar a expressão com um tempo máximo definido pelo parâmetro de *Time Out*. Quando o resolvedor SAT dá a resposta de que é possível satisfazer a expressão criada, significa que existe uma combinação de valores de sinais que testam a falha em questão. Em caso negativo, o resolvedor pode apresentar dois comportamentos possíveis: a falha pode ser abortada por não ter conseguido encontrar uma solução dentro do tempo de *Time Out* estipulado ou foi determinado pela instância SAT a impossibilidade de se satisfazer a expressão, ou seja, não existe um padrão de entrada que faça o efeito de falha se propagar

até uma saída primária (*unSAT*). Ao analisar todas as falhas, o processo é finalizado passando para a etapa de pós-processamento para criar os arquivos de saída.

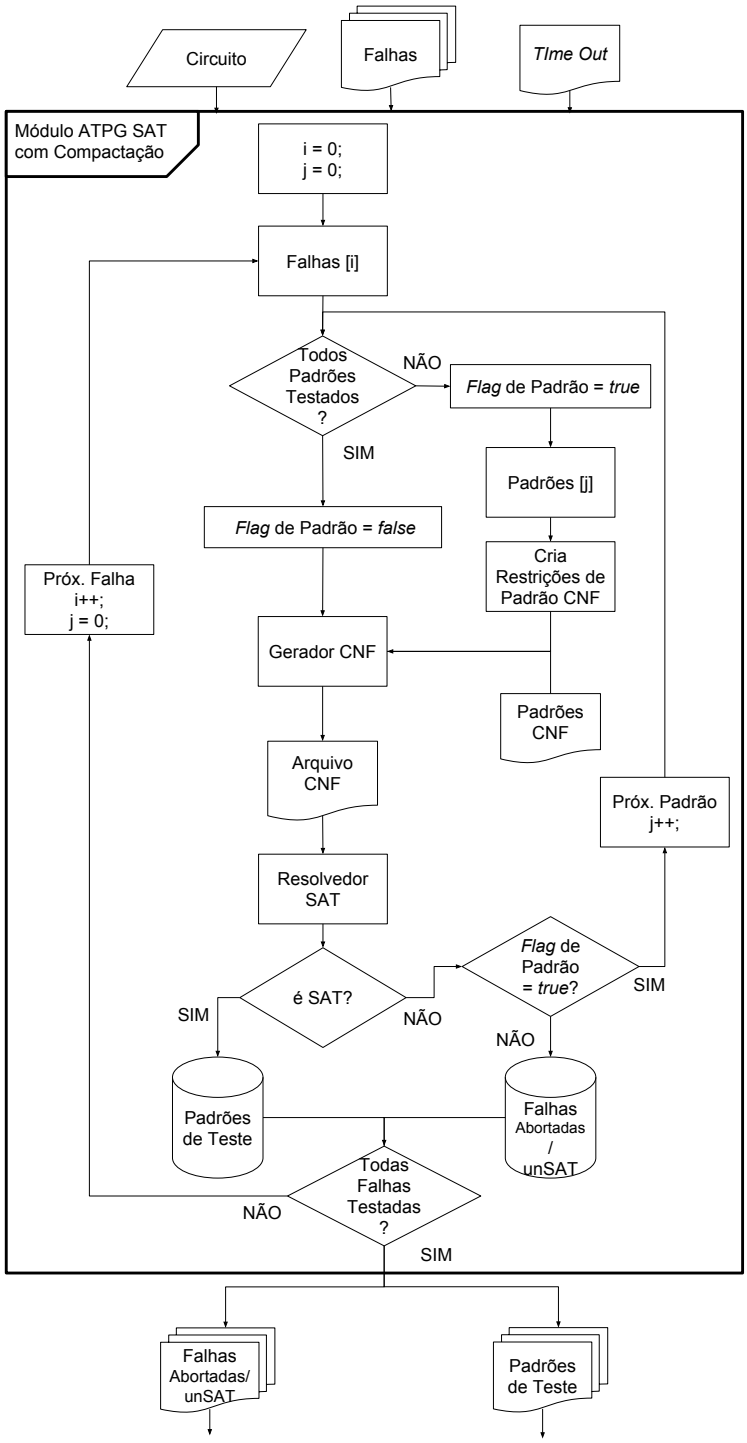


Figura 23 – Fluxograma ATPG baseado em SAT com Compactação de Padrões de Teste.

O segundo módulo ATPG baseado em SAT, representado pelo fluxograma da Figura 23, tem o intuito de aplicar uma metodologia de compactação do conjunto de padrões de teste. Ao receber o circuito e o parâmetro do método, é iniciado o laço de análise das falhas. Nessa metodologia, a primeira iteração do laço funciona igual ao ATPG

baseado em SAT clássico; a diferença acontece a partir da segunda iteração. Ao iniciar a nova iteração, para cada falha será analisado o conjunto de padrões gerados até o momento. Nesse caso, o gerador CNF, além de considerar a falha da iteração, cria uma restrição na expressão CNF para verificar se algum padrão já gerado testa a falha corrente. Ao analisar a expressão, o resolvedor SAT dá um retorno positivo se o padrão em análise consegue testar a falha corrente, e – em caso negativo – é testado o próximo padrão do conjunto até analisar todos os padrões ou até encontrar um que teste a falha. Ao encontrar um padrão do conjunto já gerado que teste a falha corrente, essa é correlacionada ao padrão já existente. Porém, ao analisar todos os padrões e nenhum deles terminar em caso positivo, será necessário tentar gerar um novo padrão de teste. O tempo de análise por falha aumenta em relação à implementação clássica, porém ao direcionar a solução via expressão CNF é possível obter conjuntos de padrões reduzidos em relação à implementação clássica.

4.2 Composição de Métodos: ATPGs Híbridos

Três classes de ATPG foram implementadas: metodologia aleatória, estrutural e a baseada em SAT. Em mãos dos métodos implementados, foram esquematizados três métodos híbridos, são eles: método aleatório estrutural com algoritmo-D, método aleatório estrutural com ATPG baseado em SAT com Compactação e método aleatório estrutural, ATPG baseado em SAT com Compactação e Algoritmo-D. Com isso, a seguir será explanada a esquematização de cada módulo híbrido desenvolvido, bem como quais os seus objetivos para a análise que é apresentada no Capítulo 5.

O objetivo foi esquematizar as melhores composições de métodos, baseando-se em análises gerais prévias. Primeiramente, o método ATPG – que é comum entre os três híbridos apresentados neste trabalho – é o método aleatório estrutural. Fundamentando-se na literatura (WANG; WU; WEN, 2009), a metodologia aleatória é uma aplicação essencial na redução de carga para métodos de geração de padrões principais por se tratar de um método ATPG de baixa complexidade, tendo o intuito de – em todos os casos – aumentar a performance final do método. De acordo com análises prévias, que são comprovadas no próximo capítulo, a abordagem estrutural para a metodologia aleatória foi escolhida por apresentar um melhor desempenho em relação à abordagem baseada em SAT.

4.2.1 Aleatório Estrutural e Algoritmo-D

Após a execução do pré-processamento, a composição de métodos começa a ser executada. O método híbrido composto pelo método aleatório estrutural e pelo algoritmo-D, representado pelo fluxograma da Figura 24, começa a sua execução recebendo a estrutura

de dados representando o circuito, o conjunto de falhas a ser analisado e os parâmetros configuráveis de acordo com os métodos utilizados na composição, limite para a abordagem aleatória e limite de *backtrack* para o Algoritmo-D.

Com esses dados, é executado o método aleatório estrutural, em seguida são produzidos dois conjuntos: o conjunto de padrões de teste e de falhas abortadas pelo método. O conjunto de falhas abortadas pelo método aleatório estrutural é utilizado como entrada para o método Algoritmo-D. Ao analisar todas as falhas restantes pelo processo, é produzido o conjunto final de falhas abortadas e padrões de teste são acrescentados ao conjunto já criado pelo método aleatório. Ao final, é feito o pós-processamento levando em conta a execução total dos dois métodos em conjunto.

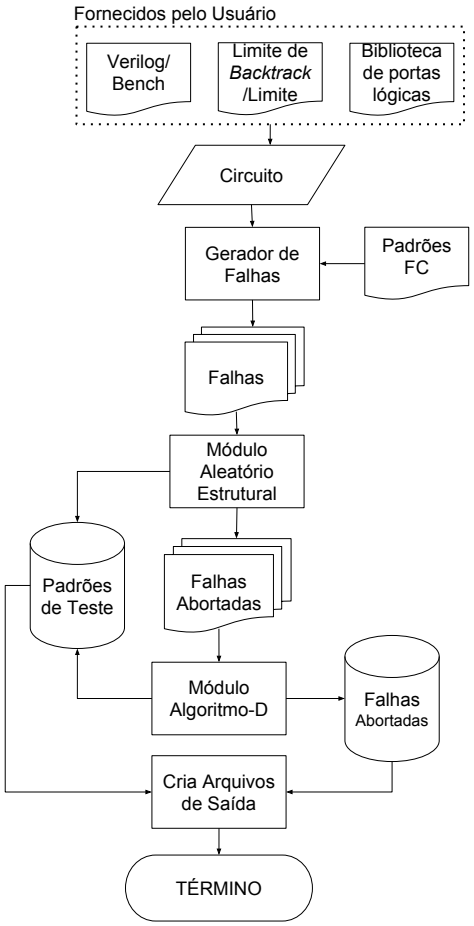


Figura 24 – Fluxograma ATPG Híbrido: Aleatório Estrutural + Algoritmo-D

4.2.2 Aleatório Estrutural e Método baseado em SAT com Compactação

A segunda composição de métodos apresentada neste trabalho é composta pelos métodos ATPG aleatório estrutural e ATPG baseado em SAT com compactação, representados pelo fluxograma da Figura 25. O método baseado em SAT com Compactação foi escolhido para a composição devido ao seu caráter de gerar um conjunto reduzido de

padrões de teste. A estruturação do método híbrido corrente segue a mesma lógica do método anterior, o qual é iniciado pela etapa de pré-processamento, seguida da geração de padrões pelo módulo de ATPG Aleatório Estrutural. Em seguida, são utilizadas as falhas abortadas pelo módulo Aleatório para processamento no módulo do método baseado em SAT com Compactação. Ao final da geração do conjunto de padrões, são criados os arquivos de saída baseados nos dados gerados pela execução total dos métodos.

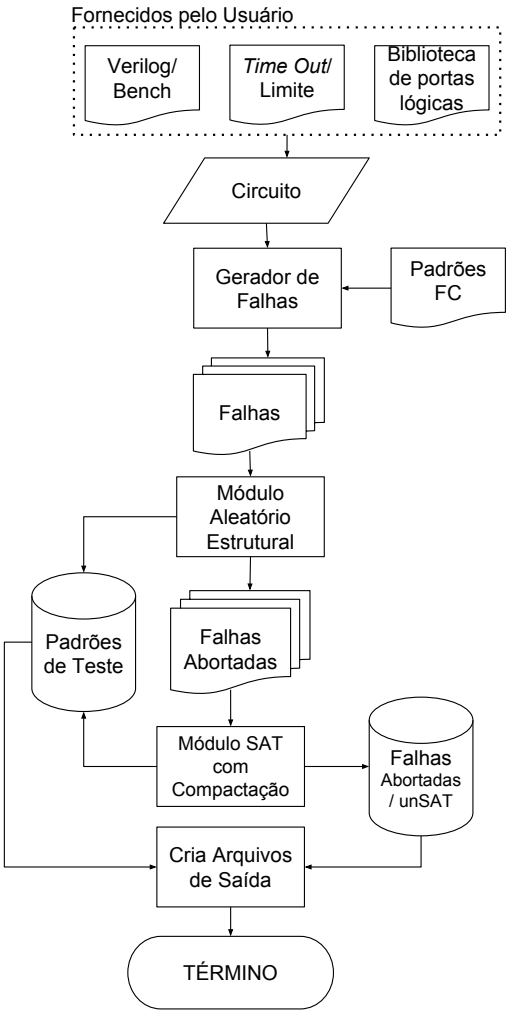


Figura 25 – Fluxograma ATPG Híbrido: Aleatório Estrutural + SAT com Compactação

4.2.3 Aleatório Estrutural, Método baseado em SAT com Compactação e Algoritmo-D

O último método híbrido proposto é composto por três métodos, um de cada classe: aleatório estrutural, baseado em SAT com compactação e Algoritmo-D. O método representado pelo fluxograma da Figura 26, começa com as estruturas definidas pelo pré-processamento, recebendo os três possíveis parâmetros configuráveis: limite para o aleatório; *Time Out* para o baseado em SAT e limite de *backtrack* para o Algoritmo-D.

A ordem de execução foi definida para ser executado o ATPG baseado em SAT anteriormente ao Algoritmo-D. Isso se justifica já que o método baseado em SAT processa mais facilmente falhas do tipo *hard-to-test* e possui a capacidade de determinar as falhas impossíveis de gerar um padrão (*unSAT*), reduzindo a carga para o método Algoritmo-D que – por sua vez – tem facilidade com falhas do tipo *easy-to-test*. Ao término da execução do método baseado em SAT é produzido um conjunto de falhas abortadas e um de falhas não possíveis de satisfazer.

O módulo Algoritmo-D é alimentado apenas com o conjunto de falhas abortadas do método baseado em SAT, desconsiderando as falhas *unSAT*. O objetivo, então, é tentar computar falhas mais difíceis com o baseado em SAT para deixar o restante que, em tese, seriam as mais fáceis para o método estrutural. O intuito do método contendo as três classes é o de explorar as melhores características de cada método.

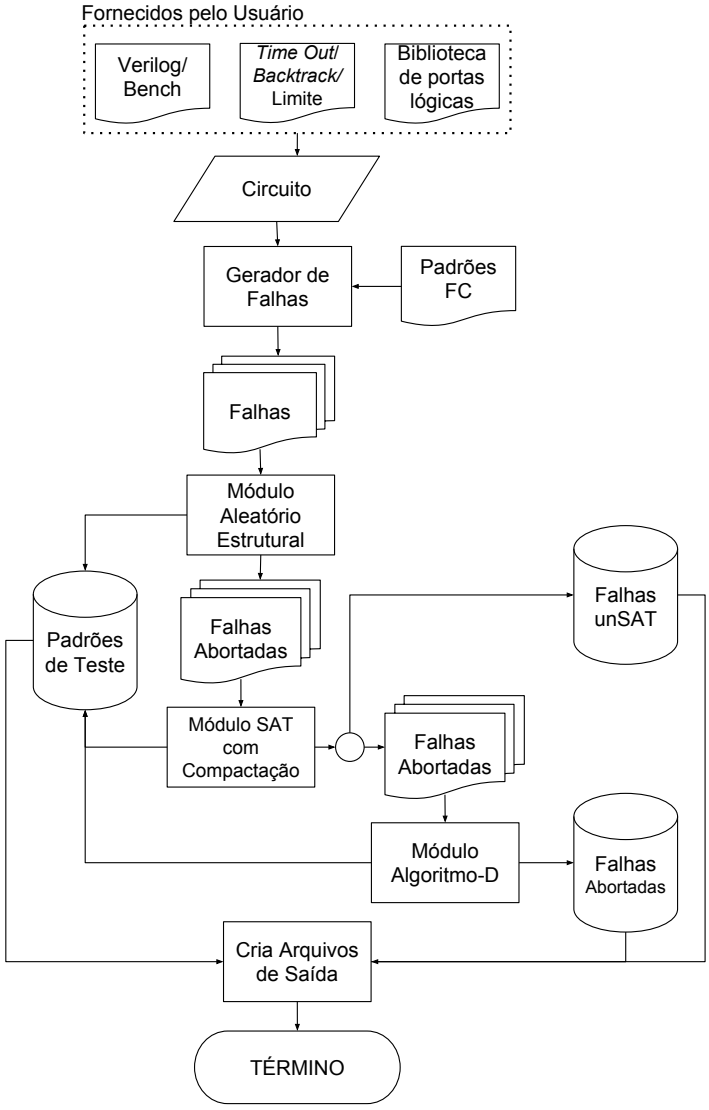


Figura 26 – Fluxograma ATPG Híbrido: Aleatório Estrutural + SAT com Compactação + Algoritmo-D

5 Resultados

Neste capítulo, será apresentada a validação dos métodos ATPGs desenvolvidos, bem como diversas análises de desempenho. Também será apresentada a métrica proposta de Figura de Mérito (*Figure of Merit* do inglês, ou FoM) e suas possíveis configurações para comparação de métodos ATPG. Em seguida, será feito um experimento para demonstrar a eficiência no uso do método de *Fault Collapsing*. Após ser feito as análises dos métodos executados de forma individual, serão realizadas as análises de desempenho utilizando os métodos híbridos..

5.1 Métrica Figura de Mérito (*Figure of Merit* - FoM)

Entre os dados de saída mais relevantes gerados por uma ferramenta ATPG, podem-se citar três: cobertura de falhas, quantidade de padrões de teste gerados e tempo de execução da ferramenta. Conforme explanado anteriormente, a cobertura de falhas determina a taxa de falhas que o conjunto gerado de padrões consegue testar, e – de acordo com o tamanho do conjunto – pode-se ter uma estimativa do tempo que será necessário para testar cada circuito. O tempo de execução representa o tempo dispendido para a geração do conjunto de padrões.

Houve a necessidade de quantificar a qualidade dos resultados de um ATPG. A qualificação de um novo método geralmente não pode ser determinada por critérios isolados, já que tais informações geradas são intrinsecamente conectadas. Por exemplo, é de pouca valia um método que gere uma cobertura de falhas próxima aos 100% utilizando um conjunto de padrões muito grande.

Com esse intuito, foi estabelecida uma métrica que relaciona esses três fatores, a fim de quantificar e comparar a qualidade de diferentes métodos. A Equação 5.1 define o cálculo da métrica *FoM*.

$$FoM = (1 - FC)^m \cdot \#P^n \cdot TE^p \tag{5.1}$$

Para avaliar o resultado de um método ATPG é necessário relacionar os fatores gerados por ele. A primeira parcela $(1 - FC)$ determina a taxa de falhas que não é coberta pelo conjunto de padrões gerado; $\#P$ representa o tamanho do conjunto de padrões de teste gerado; e TE denota o tempo de execução para alcançar os valores dos fatores anteriores. Esses três fatores devem ser minimizados e, neste caso, quanto menor o valor da Figura de Mérito, maior será considerado o desempenho do método em análise.

Cada expoente associado a um fator (expoentes m,n e p) é configurável de acordo com o que se quer avaliar no método. Por exemplo, caso se queira determinar que o tamanho do conjunto de padrões seja um aspecto mais ou menos relevante para a análise, o expoente n terá um valor maior ou menor que os outros fatores. Desta mesma forma, a configuração é análoga para os demais fatores da equação.

Nas análises realizadas neste trabalho, os fatores são configurados de acordo com o objetivo da análise em questão. Quando necessário, será calculada a Figura de Mérito com o intuito de facilitar a compreensão das análises efetuadas.

5.2 Impacto do Método *Fault Collapsing* sobre Métodos ATPG

Como abordado anteriormente, o método *Fault Collapsing* visa a redução de esforços na síntese e análise de padrões de teste (UBAR et al., 2010), influenciando diretamente no tempo de execução. A redução na quantidade de falhas a serem consideradas no processo de geração de teste é dependente de qual metodologia de relações entre falhas é tomada. O método funcional é baseado na análise da mudança da função lógica do circuito pela falha, tornando o processo mais complexo. A metodologia estrutural é a mais utilizada por ser de fácil implementação, conseguindo uma redução de 50% em média do conjunto de falhas original, segundo a literatura (AGRAWAL; PRASAD; ATRE, 2003).

Neste trabalho, a abordagem adotada é estrutural para redução do conjunto de falhas. A Tabela 5 apresenta o conjunto *benchmark* ISCAS85 (BRYAN, 1985), o qual será utilizado ao longo de todas as análises. Nessa tabela, é mostrada a quantidade de portas lógicas e sinais que compõem cada circuito, também uma coluna indicando o total de possíveis falhas do tipo *stuck-at* e, por último, uma coluna que mostra a quantidade de falhas após a aplicação do método *Fault Collapsing*.

Para comprovar o impacto do uso do *Fault Collapsing*, foi executada a aplicação clássica baseada em SAT com e sem o uso do método em questão. O gráfico da Figura 27 representa valores normalizados de quantidades de falhas a serem analisadas e tempo de execução do método com uso do método *Fault Collapsing* em relação ao método sem uso de *Fault Collapsing*. É visto que os valores da quantidade de falhas e do tempo de execução obtidos ao utilizar *Fault Collapsing* representam por volta de 50% dos valores originais quando não é utilizado o método. Ao reduzir falhas consideradas logicamente equivalentes do conjunto, o tempo de execução é reduzido na mesma proporção. Portanto, através do experimento prático a afirmativa que consta na literatura foi reafirmada. Pode-se concluir, então, que o método de *Fault Collapsing* é essencial para redução de carga sobre métodos de geração de padrões de teste, já que ao reduzir falhas logicamente equivalentes a qualidade do teste se mantém sem perda na cobertura de falhas.

Tabela 5 – Informações do *benchmark* ISCAS85 (BRYAN, 1985).

Circuitos	#Portas Lógicas	#Sinais	#Falhas	#Falhas FC
c17	6	11	34	22
c432	160	196	864	524
c499	202	243	998	758
c880	383	443	1760	942
c1355	546	587	2710	1574
c1908	880	913	3816	1879
c2670	1193	1350	5340	2747
c3540	1669	1719	7080	3428
c5315	2307	2485	10630	5350
c6288	2416	2448	12576	7744
c7552	3512	3718	15104	7550

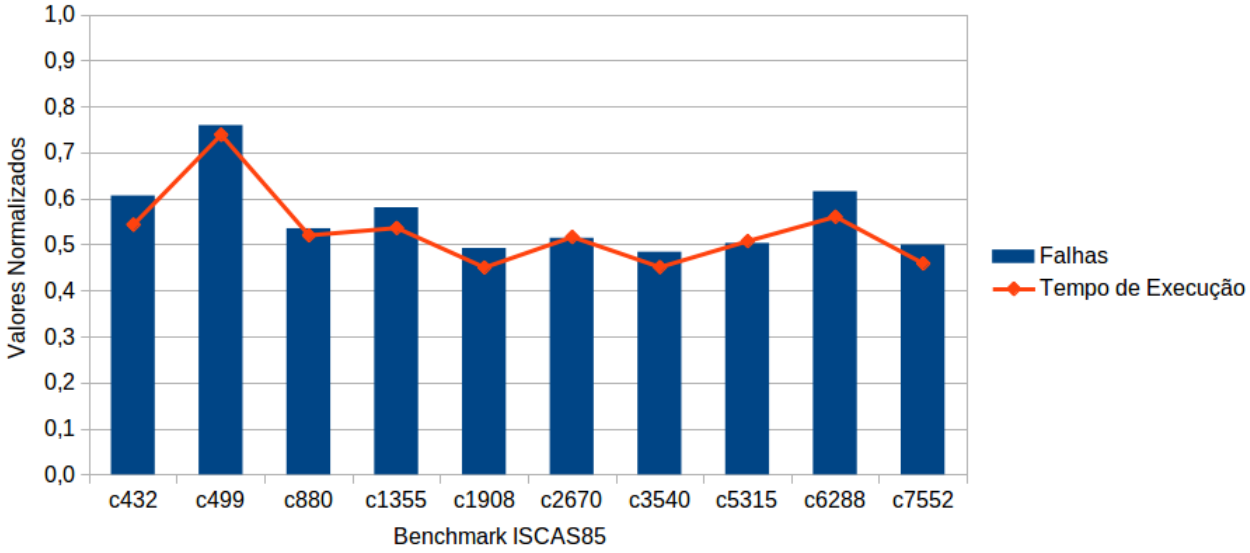


Figura 27 – Valores normalizados da aplicação clássica baseada em SAT com uso de *Fault Collpasing* em relação ao método sem uso de *Fault Collapsing* para os dados de conjunto de falhas a serem analisadas e tempo de execução da aplicação.

5.3 ATPGs Aleatórios

Como dito no Capítulo 4, a abordagem aleatória não é a melhor alternativa para geração de padrões de teste. No entanto, métodos aleatórios possuem baixa complexidade computacional. Nesse contexto, torna-se uma boa alternativa para a interação entre outros métodos com o objetivo de redução de carga.

O experimento consistiu em executar os dois métodos (Aleatório baseado em SAT e Aleatório Estrutural) variando o parâmetro de limite de respostas sem sucesso de cobertura para validação do método. Para uma maior avaliação dos métodos aleatórios, foram

executadas dez rodadas para cada valor de limite no circuito c432. O intuito é a verificação do comportamento de padrões de cobertura de falhas, conjunto de padrões de teste e tempo de execução. As análises feitas a seguir são baseadas nos dados obtidos presentes nos Apêndices B.2 e B.1.

A Tabela 6 apresenta valores de média, desvio padrão e a significância do desvio padrão sobre a média em dez rodadas de execução do circuito c432. Como esperado, é possível perceber que, ao aumentar o limite de parada do método, a média de padrões de teste e cobertura de falhas aumentam. Porém, ao verificar os valores de desvio padrão e a relação do desvio sobre a média, é possível observar realmente seu comportamento. Por meio da razão do desvio padrão sobre a média, é visto que, ao aumentar o limite de parada do método, a significância do desvio padrão cai drasticamente. Tal comportamento indica que quanto maior o limite, menor será a variação de valores.

Com isso, pode-se concluir que seu comportamento aleatório cai em função do aumento do limite. É importante salientar que o impacto da redução da aleatoriedade está no tamanho do conjunto de padrões e não no conteúdo do conjunto, haja vista que a geração dos padrões é de forma aleatória.

Tabela 6 – Dados de tamanho de conjunto de padrões, cobertura de falhas e tempo de execução para experimento do Método Aleatório Estrutural para 10 rodadas para o circuito c432.

	Padrões			Cobertura			Tempo (s)		
Limite	Méd.	Desv. Pad.	Desv./Méd.	Méd.	Desv. Pad.	Desv./Méd.	Méd.	Desv. Pad.	Desv./Méd.
1	14,1	8,32	0,59	52,16%	18,71%	0,36	1,49	0,52	0,35
2	25,2	7,43	0,29	73,72%	9,09%	0,12	2,25	0,53	0,23
4	42,0	7,81	0,19	84,56%	4,63%	0,05	3,48	0,61	0,18
8	55,7	6,53	0,12	90,59%	3,18%	0,04	5,06	0,71	0,14
16	67,3	6,05	0,09	93,93%	2,39%	0,03	6,76	1,08	0,16
32	73,7	2,57	0,03	96,24%	1,36%	0,01	10,50	2,66	0,25

O tempo de execução possui um comportamento diferente da cobertura de falhas e o tamanho do conjunto de teste. Pode-se perceber que, mesmo aumentando o limite do método aleatório, a significância do desvio padrão sobre a média se mantém sem grande variação. Esse comportamento indica que o tempo de execução não é tão afetado pela aleatoriedade do processo.

A primeira análise se detém na avaliação de limites a serem utilizados. Nesse viés, é avaliado o comportamento temporal para diferentes circuitos, como visualizado na Figura 28 (eixo y em escala logarítmica). É visto que, independente da estrutura do circuito, o comportamento temporal cresce linearmente. Entretanto, é visto que quanto mais complexo o circuito, mais tempo é necessário para simular o padrão gerado aleatoriamente. Por exemplo, o circuito c7552 possui uma média de 20 vezes mais portas lógicas e sinais que o circuito c432, apresentando um aumento de aproximadamente 1000 vezes no tempo de execução. Levando em consideração a crescente do tempo e o fato de que quanto maior

for o limite, menor a variação dos demais fatores, não é interessante a utilização de limites altos. A pouca variação dos parâmetros acaba por não compensar o aumento do tempo de execução. Ao colocar limites altos, o objetivo de métodos aleatórios é comprometido, uma vez que o intuito é oferecer um método rápido para aceleração de métodos ATPGs os quais serão executados posteriormente.

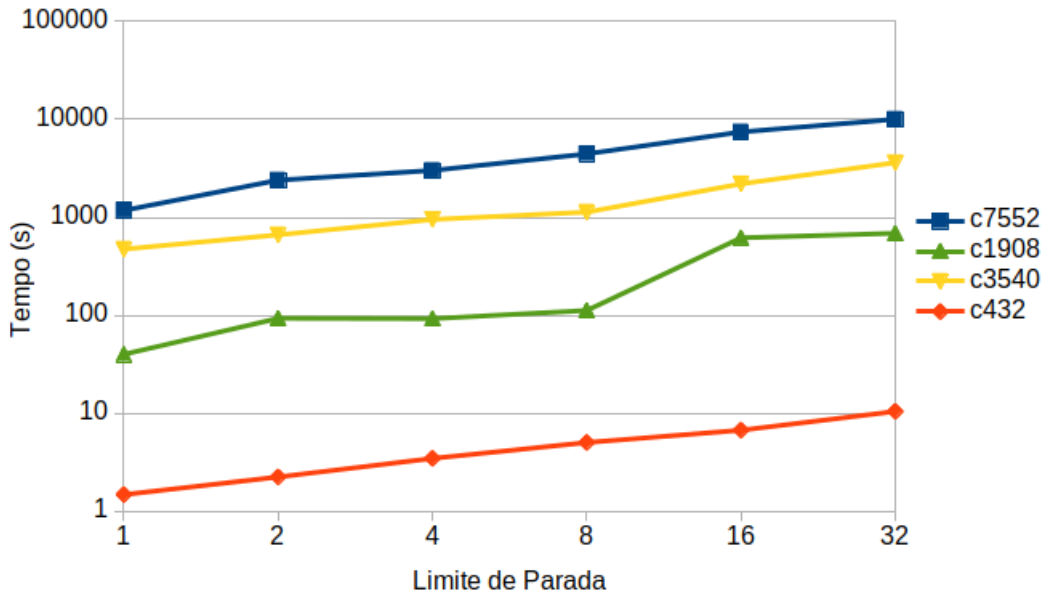


Figura 28 – Comportamento do tempo de execução de acordo com o limite imposto para o Aleatório Estrutural.

As análises feitas até aqui com o Aleatório Estrutural são análogas para o Aleatório baseado em SAT, tendo em vista que o que muda é o modo como os padrões gerados são simulados no circuito. Com isso, torna-se interessante verificar qual das metodologias é a mais adequada para simulação de padrões. Utilizando como exemplo, temos o comportamento temporal do circuito c7552 representado na Figura 29. É visto que quanto maior o limite, menor se torna a diferença no tempo de execução entre as duas metodologias e, no seu pior caso, o método estrutural é 50% mais rápido. Uma vez que o intuito é a utilização de limites intermediários, pode-se concluir que para soluções aleatórias, metodologias estruturais para simulação de circuitos são mais eficientes que as baseadas em SAT, já que a diferença temporal é substancialmente grande.

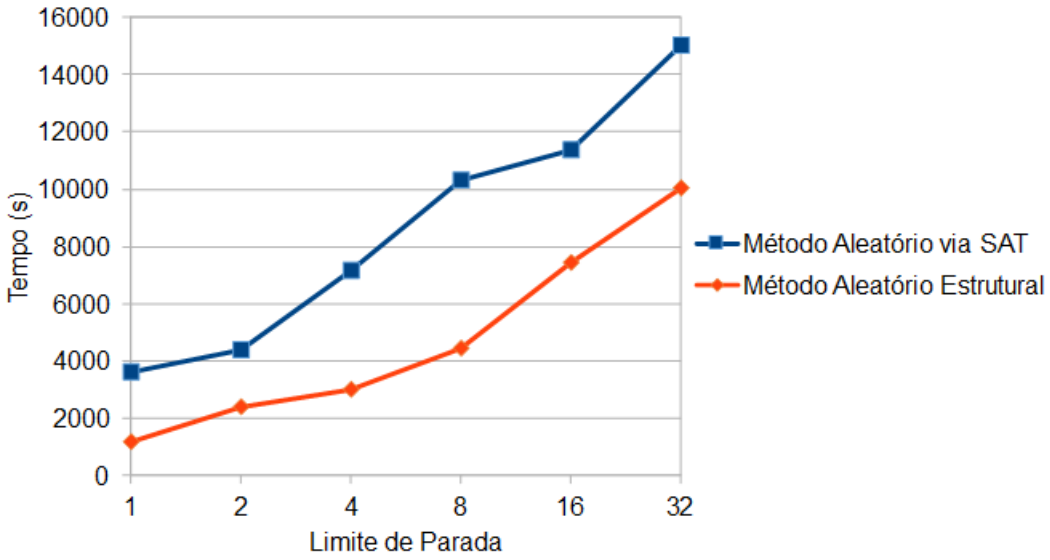


Figura 29 – Tempo de execução para o circuito c7552 utilizando os métodos Aleatório Estrutural e baseado em SAT.

5.4 Método Estrutural Clássico - Algoritmo-D

O método Algoritmo-D (ROTH, 1966) foi escolhido por ser a base para todos aprimoramentos subsequentes em ATPG estrutural e demonstrar claramente limitações de metodologias estruturais para falhas difíceis de serem testadas. A seguir, serão apresentadas características do método e análises baseadas na validação do método presente no Apêndice B.3.

Para a validação do Algoritmo-D foi utilizado o limite de quantidade de *backtracks* igual a 1. Essa configuração foi utilizada devido à limitação de alto tempo de execução, tornando possível validar a maioria dos circuitos do conjunto *benchmark*. É possível verificar na Tabela 7 que com pouco espaço para refazer escolhas, a taxa de cobertura de falhas é relativa dependendo da estrutura do circuito.

Em alguns dos casos, como é nos circuitos c499,c1355 e c1908, a taxa de cobertura se torna baixa devido a estrutura ser do tipo ECAT (*Error-correction-and-translation*). Circuitos ECAT são caracterizados por conterem muita lógica XOR na sua estrutura. Portas do tipo XOR são péssimas para esse tipo de aplicação, pois não é possível de ser aplicada nenhuma operação de implicação sobre elas. Circuitos com uma maior profundidade lógica tendem a ter menor controlabilidade e observabilidade, o que faz com que se torne mais difícil de gerar padrões. Consequentemente, o tempo de processamento se torna maior. Também, quanto mais complexa e maior é uma estrutura, mais decisões pode se tomar, fazendo com que a probabilidade de gerar um padrão com um limite baixo de *backtrack* seja reduzida.

Tabela 7 – Validação para *Backtrack=1* no Algoritmo-D.

Circuitos	#Padrões	Cobertura	Tempo (s)
c17	13	100,00%	0,48
c432	200	95,42%	179,48
c499	76	35,88%	241,06
c880	463	100,00%	978,03
c1355	50	17,41%	6.993,63
c1908	375	56,47%	25.859,18
c2670	928	80,49%	71.001,22
c3540	1076	73,37%	114.437,75
c5315	2031	88,88%	494.336,12

Uma análise interessante de ser feita é verificar a influência da quantidade de *backtracks* na execução do Algoritmo-D. A Tabela 8 apresenta dois casos distintos: o circuito c432, cujo método possui maior facilidade de geração, enquanto o c499 apresenta o caso de limitação sendo um exemplo de estrutura ECAT. É visto que quando é utilizado para circuitos com portas lógicas básicas como NAND e NOR, o aumento do tempo em questão do número de *backtracks* não é tão discrepante como é o caso de circuitos com lógicas mais complexas.

O mesmo pode ser visto no valor da cobertura de falhas, em que circuitos com portas lógicas mais simples conseguem atingir valores mais altos de cobertura como é o caso do c432, enquanto no caso do circuito c499 não se consegue atingir nem 40% de cobertura de falhas. O circuito c432 aumentou $3,5\%$ de cobertura de falhas atingindo 99% de cobertura com um aumento de apenas 24% no tempo de execução, ao passo que o circuito c499 obteve um aumento de 2% de cobertura de falhas sem atingir nem 40% de cobertura com um aumento de 6 vezes no tempo de execução quando usado *backtrack* igual a 32 em comparação quando utilizado o parâmetro igual à 1.

Tabela 8 – Experimento da influência do limite de *backtrack* na cobertura de falhas e tempo de execução

	c432		c499	
Backtrack	Cobertura	Tempo (s)	Cobertura	Tempo (s)
1	95,42%	173,15	35,88%	250,34
2	97,90%	181,53	35,88%	257,19
4	98,47%	184,96	35,88%	295,21
8	98,47%	190,69	36,94%	523,24
16	98,47%	200,17	36,94%	670,68
32	99,05%	214,79	37,99%	1.502,32

Pode-se concluir com esta análise que a complexidade de funções lógicas e tamanho do circuito afetam diretamente o desempenho de algoritmos estruturais, principalmente

em se tratando do Algoritmo-D. Essa desvantagem é devido a uma maior quantidade de escolhas que podem ser tomadas. Tal característica de métodos estruturais torna necessária a elaboração de heurísticas para contornar esse problema, como é o caso das implementações subsequentes PODEM (GOEL, 1981) e FAN (FUJIWARA; SHIMONO, 1983). Quanto maior é o número de escolhas a serem tomadas dentro de um circuito, menor é a probabilidade de encontrar um padrão de teste adequado com um baixo número de *backtracks*, o que se reflete em uma baixa cobertura de falhas e um alto tempo de execução.

5.5 ATPGs Baseados em SAT

Conforme dito no Capítulo 4, foram desenvolvidos dois métodos baseados em SAT. Um método clássico e um outro utilizando uma abordagem que visa à compactação de padrões. A validação e análise se detêm à demonstração das principais características da aplicação SAT e ao impacto da exploração de meios de guiar a aplicação SAT através da construção de fórmulas CNF. A validação dos dois métodos implementados foi realizada variando o parâmetro de *Time Out*. Os dados obtidos podem ser encontrados nos Apêndices B.5 e B.7.

O resolvedor SAT aplicado a ATPGs foi proposto como uma melhoria para métodos estruturais que começaram a apresentar problemas de escalabilidade. Por intermédio de análises práticas, é possível demonstrar as principais características que um ATPG baseado em SAT clássico possui.

A primeira característica apresentada é a capacidade de verificar a cobertura de falhas máxima que uma estrutura pode alcançar. A Tabela 9 demonstra a progressão de falhas abortadas, falhas *unSAT* e cobertura de falhas, conforme é aumentado o valor de *Time Out*. Falha abortada é aquela com a qual não foi possível gerar um padrão dentro do tempo estipulado ao resolvedor. Paralelamente, uma falha *unSAT* é aquela que, dentro do tempo estipulado, o resolvedor SAT definiu-a como impossível de ser satisfeita, ou seja, não existe um padrão que teste a falha em questão.

A característica de definir quais falhas são impossíveis de serem satisfeitas determina a capacidade da ferramenta de apresentar a cobertura de falhas máxima possível, dada uma estrutura. Dessa forma, pode-se dizer que quando o conjunto de falhas abortadas se torna vazio a cobertura de falhas converge ao seu valor máximo. É visto que ao atingir $1s$ de *Time Out*, o método baseado em SAT consegue saturar a cobertura de falhas, com exceção do circuito c6288. Essa característica torna-se muito interessante, pois com o valor máximo de cobertura de falhas definido, permite-se verificações e comparações de cobertura de falhas obtidas por métodos de diversas metodologias.

Tabela 9 – Falhas abortadas, *unSAT* e Cobertura de Falhas em relação ao *Time Out*.

	0,01s			0,1s			1s		
Circuitos	#Abort.	#unSAT	Cobertura	#Abort.	#unSAT	Cobertura	#Abort.	#unSAT	Cobertura
c432	16	4	96,18%	0	4	99,24%	0	4	99,24%
c499	14	8	97,10%	0	8	98,94%	0	8	98,94%
c880	20	0	97,88%	0	0	100,00%	0	0	100,00%
c1355	39	8	97,01%	0	8	99,49%	0	8	99,49%
c1908	274	7	85,05%	0	7	99,63%	0	7	99,63%
c2670	599	108	74,26%	0	115	95,81%	0	115	95,81%
c3540	1065	88	66,37%	10	127	96,00%	0	131	96,18%
c5315	1959	43	62,58%	1	59	98,88%	0	59	98,90%
c6288	6862	2	11,36%	999	3	87,06%	44	7	99,34%
c7552	6316	96	15,07%	8	130	98,17%	0	131	98,26%

Outra característica importante que tornou aplicações ATPG baseada em SAT uma alternativa para métodos estruturais, como o Algoritmo-D apresentado anteriormente, é o baixo tempo de execução. O gráfico da Figura 30 mostra o tempo de execução para três aplicações: Algoritmo-D (com *Backtrack = 1*), Aleatório Estrutural (com *Limite = 32*) e SAT Clássico (com *Time Out = 1s*). A escolha dos dados com as configurações descritas acima foi feita baseada no melhor cenário de cobertura de falhas alcançado para cada método dentro de suas limitações.

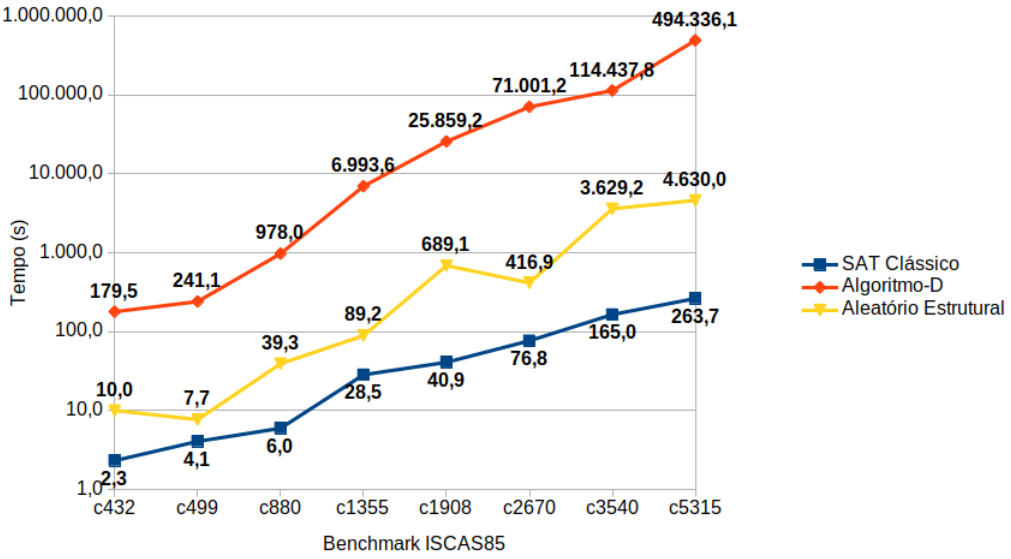


Figura 30 – Comparação do Comportamento temporal para ATPGs SAT Clássico, Algoritmo-D e Aleatório Estrutural.

A partir da Figura 30, em escala logarítmica, é possível ver que o ATPG baseado em SAT clássico obteve o melhor desempenho de tempo de execução em relação ao Algoritmo-D e ao ATPG Aleatório. Sendo até 1900 vezes mais rápido que o Algoritmo-D implementado e 18 vezes mais rápido que o Aleatório Estrutural para o circuito c5315. Entretanto, a maior desvantagem da aplicação clássica é o alto número de padrões de teste. Apesar da alta performance, conseguindo convergir para o valor máximo de cober-

tura de falhas em um baixo custo de tempo de execução, um conjunto muito grande de padrões de teste compromete a etapa de teste.

O gráfico da Figura 31, em escala logarítmica, demonstra a quantidade de padrões de teste gerados para o conjunto de *benchmark* utilizado. Não obstante ao ganho de desempenho em questão temporal e cobertura de falhas, o método baseado em SAT Clássico apresenta um aumento significativo no número de padrões de teste em relação às outras implementações em comparação. O método baseado em SAT Clássico apresenta um aumento em 90% e 60% em média para o Aleatório Estrutural e Algoritmo-D, respectivamente.

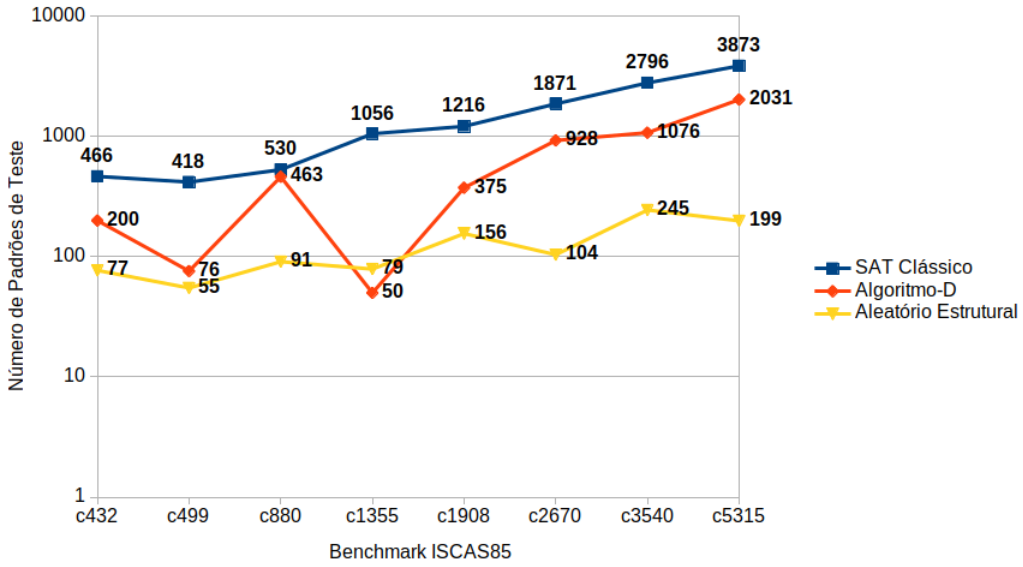


Figura 31 – Comparação da Quantidade de Padrões de teste gerados pelos ATPGs SAT Clássico, Algoritmo-D e Aleatório Estrutural.

É possível contornar a desvantagem apresentada na aplicação clássica por meio da manipulação de expressões CNF. A segunda implementação de um ATPG baseado em SAT possui uma metodologia que visa a compactação do conjunto de padrões de teste, direcionando a resposta do resolvedor SAT por via de uma estruturação diferente da expressão CNF.

A comparação entre os tamanhos dos conjuntos de padrões de teste pelas duas aplicações SAT é representada no gráfico da Figura 32. Os dados utilizados nessa comparação e nas subsequentes são os obtidos utilizando o parâmetro de *Time Out = 1s*, o qual para as duas aplicações é o valor de *Time Out* em que a cobertura de falhas converge para o seu valor máximo possível.

Pode ser visto, na Figura 32, que ao utilizar uma nova metodologia de modelagem CNF, a solução baseada em SAT com Compactação atingiu uma média de redução de 89% no tamanho do conjunto de padrões. Porém, ao criar mais expressões CNF, o tempo

de execução aumenta proporcionalmente à redução de padrões.

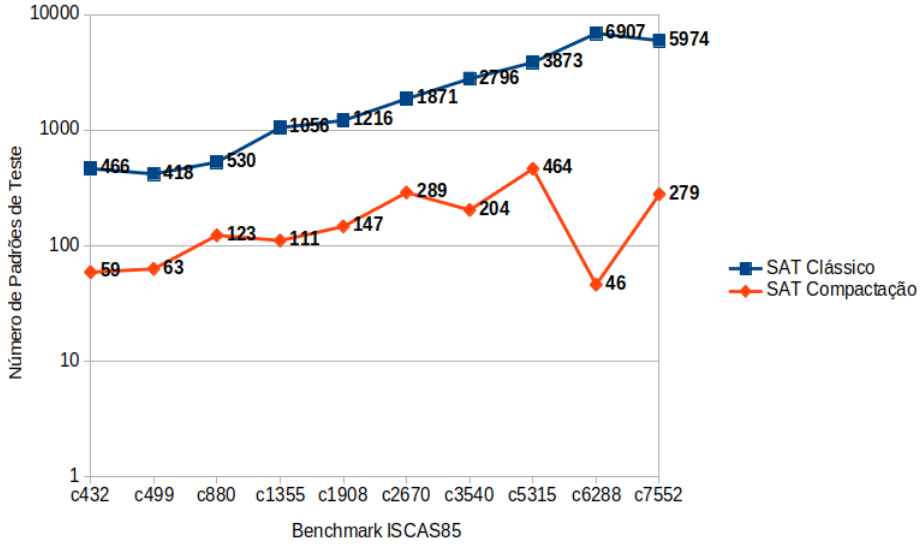


Figura 32 – Comparação da Quantidade de Padrões de teste gerados pelos ATPGs SAT Clássico e SAT com Compactação.

O gráfico da Figura 33 mostra a comparação do comportamento temporal para as duas soluções baseadas em SAT implementadas. É identificado que a redução do conjunto de padrões de teste tem o custo no aumento do tempo de execução. Ao utilizar a metodologia baseada em SAT com Compactação, o tempo de execução apresentou um aumento médio de 89% em relação à aplicação clássica.

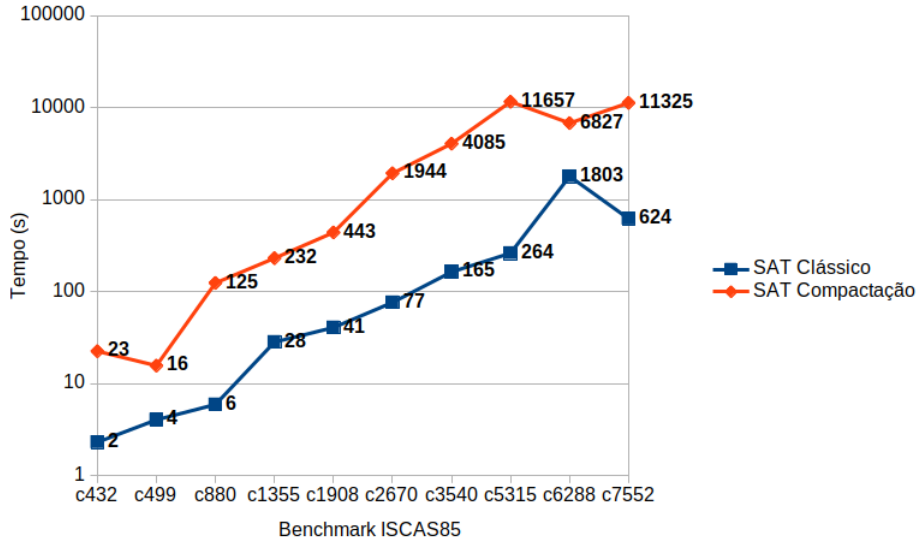


Figura 33 – Comparação do Comportamento temporal para ATPGs SAT Clássico e SAT com Compactação.

Pode-se concluir, então, que a redução do conjunto de padrões e o aumento no tempo de execução ao aplicar a metodologia de compactação proposta neste trabalho são

proporcionais. Porém, os dois referidos fatores possuem impactos diferentes no processo de teste de circuitos integrados. O tempo de execução para um método ATPG qualquer está limitado apenas à execução necessária para a obtenção do conjunto de padrões, ao passo que o tamanho do conjunto de padrões impacta diretamente no tempo de teste de uma linha de produção de sistemas integrados. Nesse sentido, a Figura de Mérito é configurada baseado nesta afirmação.

A relevância da redução do conjunto de padrões sobre o aumento do tempo de execução é representada no gráfico da Figura 34. O gráfico em questão mostra a comparação da Figura de Mérito configurada dando mais significância ao conjunto de padrões e uma menor relevância ao tempo de execução ($(1 - FC) \cdot \#P^2 \cdot TE^{\frac{1}{2}}$), em que quanto menor o valor da métrica utilizada, maior será o desempenho do método em questão.

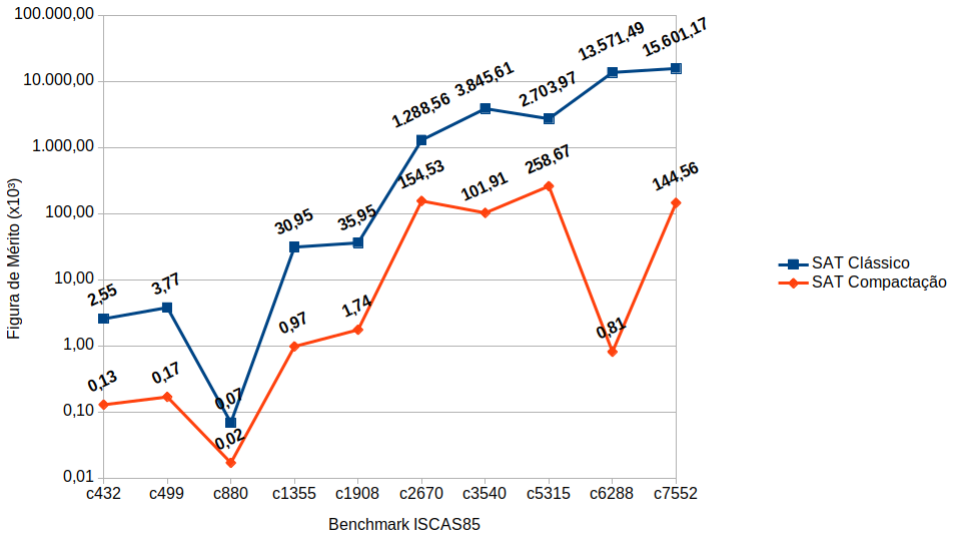


Figura 34 – Comparação da Figura de Mérito para ATPGs SAT Clássico e SAT com Compactação. $((1 - FC) \cdot \#P^2 \cdot TE^{\frac{1}{2}})$

É visto que a redução do conjunto de padrões compensa o aumento de tempo de execução, demonstrando que quanto melhor feita a modelagem CNF do problema de geração de teste, melhor será o desempenho de ferramentas baseadas em SAT. Ao ser comparada às demais implementações (vide gráfico da Figura 35), a metodologia baseada em SAT com Compactação apresenta um ganho de performance, à exceção em poucos casos do Aleatório Estrutural. Tal comportamento é justificável pelo método aleatório ser um algoritmo ATPG com baixa complexidade, podendo alcançar resultados relevantes em um tempo de execução menor.

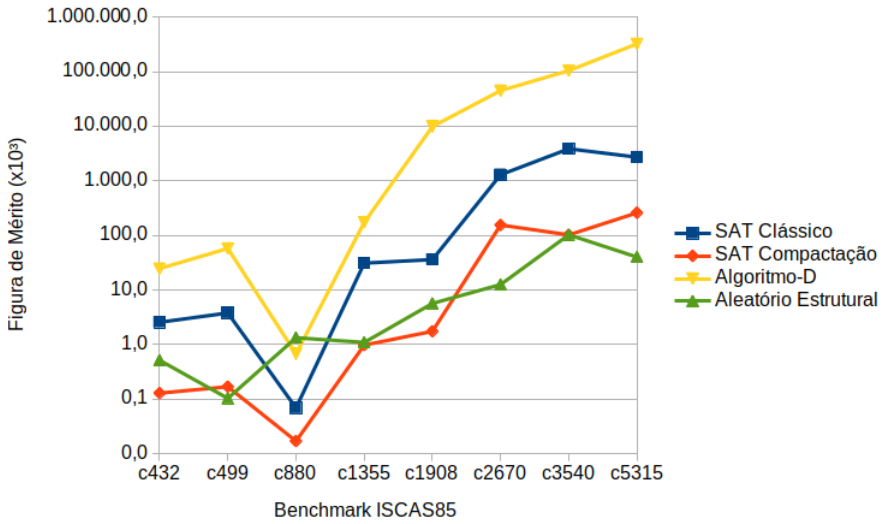


Figura 35 – Comparação da Figura de Mérito para ATPGs implementados. $((1 - FC) \cdot \#P^2 \cdot TE^{\frac{1}{2}})$

5.6 Composição de Métodos - ATPGs Híbridos

Até então foram abordadas análises e validações dos métodos executados de forma individual. Pode-se inferir, com base nas análises anteriores, que os métodos utilizados não são perfeitos, cada um possui vantagens e desvantagens. Porém, é possível amenizar os lados negativos de cada método se feita uma combinação entre eles: métodos ATPGs que combinam diversas metodologias em conjunto conhecidos como ATPGs híbridos. Esses têm por objetivo explorar características positivas de cada método, a fim de realizarem a otimização da geração de padrões de teste.

Para compor métodos híbridos, deve-se levar em consideração as características de cada método para uma melhor interação, bem como uma parametrização equilibrada. Nestas análises, serão utilizados três métodos ATPG implementados: Aleatório Estrutural, Algoritmo-D e SAT com Compactação. Os métodos para a composição foram escolhidos baseados no desempenho nas análises anteriores, ou seja, o Aleatório Estrutural e o SAT com Compactação foram escolhidos por se sobressaírem em relação ao Aleatório baseado em SAT e ao ATPG clássico baseado em SAT, enquanto o Algoritmo-D foi escolhido por ser o representante de ATPG estrutural neste trabalho.

Após a escolha dos métodos a serem utilizados para a composição, deve ser definido como irão interagir, ou seja, a estruturação de cada método híbrido. Para este trabalho foram analisados três métodos híbridos: Aleatório Estrutural com Algoritmo-D, Aleatório Estrutural com SAT com Compactação e uma terceira alternativa contendo os três métodos, Aleatório Estrutural, SAT com Compactação e Algoritmo-D, nessa respectiva ordem.

Após escolher como os métodos irão interagir, é importante saber como configurar os parâmetros de cada método para um melhor aproveitamento. No método aleatório foram utilizados limites de paradas intermediários na maioria dos casos, de acordo com a análise feita na Seção 5.3. O método Algoritmo-D foi utilizado *backtrack* igual a 1 e para o método baseado em SAT com Compactação valores inferiores a 1s de *Time Out*. Como o Algoritmo-D e o SAT com Compactação são bons instrumentos para falhas fáceis e difíceis, respectivamente, ao configurar seus parâmetros com valores reduzidos, direcionam-se os métodos para que sejam induzidos a processarem as falhas que cada um tenha mais facilidade. A seguir são apresentadas as análises referentes aos métodos híbridos.

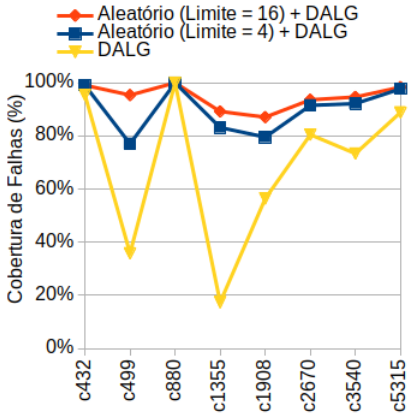
5.6.1 Aleatório Estrutural e Algoritmo-D

A primeira implementação híbrida que será abordada será o método composto pelo Aleatório Estrutural e Algoritmo-D. Por esse último possuir desvantagens – como dificuldade de geração de padrões para circuitos com portas lógicas mais complexas e o alto tempo de execução – torna-se interessante a implementação de um método híbrido para contornar tais problemas. O método Aleatório Estrutural com o Algoritmo-D foi estruturado para fins de redução de carga sobre o Algoritmo-D.

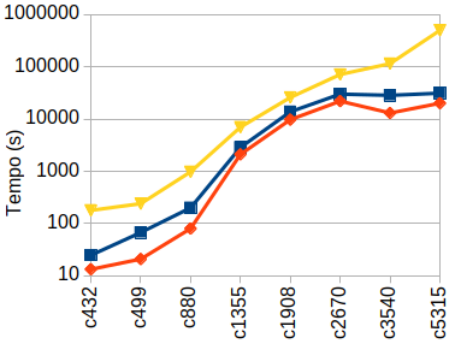
O experimento consistiu na execução e validação do método híbrido mantendo o valor de *backtrack = 1* e variando o método aleatório com limite de parada de 4 e 16. O valor de *backtrack* dá continuidade à mesma lógica anterior: realizar a geração do Algoritmo-D o mais rapidamente. A variação do limite de parada se dá por verificar a configuração em dois cenário diferentes, um quando o nível de aleatoriedade do método é alto e outro quando a aleatoriedade é reduzida. Os dados gerados pela execução do método que serão base para as análises nesta subseção estão presentes no Apêndice B.8.

A parte inicial da análise é feita comparando o método híbrido corrente com o método Algoritmo-D. O objetivo desta primeira análise é verificar o impacto do uso de um método aleatório sob o método Algoritmo-D. Os gráficos que ilustram a análise podem ser vistos na Figura 36.

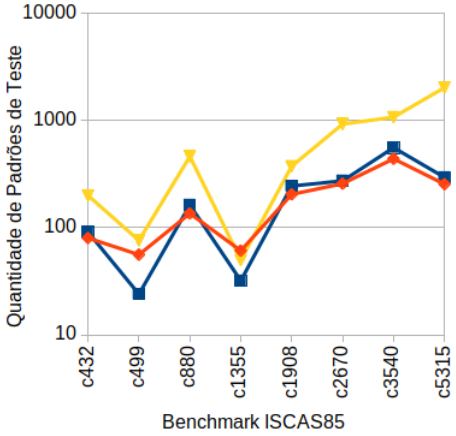
Analisando os três gráficos que comparam a cobertura de falhas (Figura 36a), tempo de execução (Figura 36b) e padrões de teste (Figura 36c), pode ser compreendido que a metodologia híbrida obteve os melhores resultados nos três cenários. No caso da cobertura de falhas, o maior impacto está nos circuitos onde o Algoritmo-D sozinho possui muita dificuldade na geração de padrões, como são os casos dos circuitos c499, c1355 e c1908. Comparando com o pior caso do Algoritmo-D, a metodologia híbrida aumentou aproximadamente $71,8\%$ e $65,76\%$ na cobertura de falhas para limites 4 e 16, respectivamente. Já para o melhor caso do Algoritmo-D, o método híbrido melhorou $3,6\%$ na cobertura de falhas.



(a) Comparação de Cobertura de Falhas.



(b) Comparação de Tempo de Execução.



(c) Comparação de Quantidade de Padrões de Teste.

Figura 36 – Comparação entre método híbrido Aleatório + DALG com DALG.

Para o tempo de execução, o método híbrido apresentou em média uma redução de 85% e 90%, para limites 4 e 16 respectivamente. Esse comportamento é devido à redução de carga sobre o Algoritmo-D, fazendo com que o processo se torne mais rápido. Por fim, o método híbrido corrente apresentou também uma redução na quantidade de padrões de teste. Tal resultado é devido ao comportamento não determinístico do método Aleatório, já que é possível para o método gerar padrões de teste com uma maior cobertura em comparação à metodologia do Algoritmo-D.

No entanto, os valores percentuais apresentados nesse início de subseção e os posteriores que serão apresentados podem apresentar variações nos resultados devido ao método aleatório. O que se deve perceber e ser analisado é o comportamento ao se utilizar o método híbrido, pois mesmo com pequenas variações o comportamento apresentado tende a se manter.

A última análise feita nesta seção é a comparação da Figura de Mérito $((1 - FC) \cdot$

$\#P^2 \cdot TE^{\frac{1}{2}}$). Para esta análise foram adicionados dois outros métodos, SAT Clássico e SAT com Compactação, conforme ser visto na Figura 37. É perceptível que as duas metodologias baseadas em SAT possuem uma performance melhor que o Algoritmo-D. Porém, ao estruturar o método utilizando o Aleatório em conjunto, foi possível concluir que o método híbrido corrente obteve resultados melhores que o SAT Clássico; todavia, ainda permanece com um desempenho menor que o método baseado em SAT com Compactação.

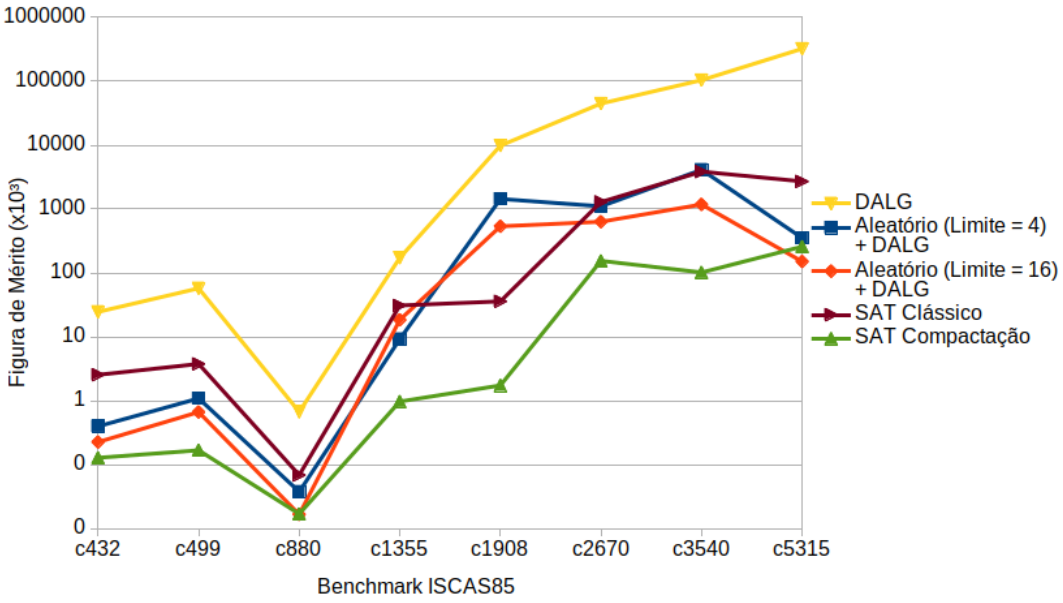


Figura 37 – Comparação da Figura de Mérito para Aleatório + DALG e demais implementações. $((1 - FC) \cdot \#P^2 \cdot TE^{\frac{1}{2}})$

Outro ponto interessante nesta análise, é visto que o método híbrido – ao utilizar limite igual a 16 – resultou em um desempenho melhor que usando limite igual a 4. Esse comportamento aparenta ser o inverso do que foi analisado anteriormente, porém é totalmente justificável.

Ao analisar de forma individual o método aleatório, a desvantagem do aumento do limite de parada é relacionada ao aumento do tempo de execução por uma pequena variação nos demais fatores, o que o faz não compensar. Porém, como para o Algoritmo-D o maior ponto negativo também é o alto tempo de execução e levando em consideração também que é mais lento que o método aleatório, o pequeno ganho na cobertura de falhas e a redução de carga maior o fazem ser mais compensatório nesse cenário de comparação.

5.6.2 Aleatório Estrutural e ATPG baseado em SAT com Compactação

A segunda implementação híbrida analisada é a composta pelo método Aleatório Estrutural com o ATPG baseado em SAT com Compactação. O método baseado em SAT com Compactação foi implementado com o intuito de reduzir o tamanho do conjunto de teste em comparação à metodologia clássica. Para que se pudesse reduzir o conjunto de

teste, foi necessário um maior tempo de processamento. Dessa maneira, é possível, então, diminuir o impacto do tempo de execução acrescentando o método Aleatório Estrutural para uma redução de carga sobre o método baseado em SAT.

A validação do método híbrido corrente se deu na variação do parâmetro de limite de parada para o método aleatório e o parâmetro de *Time Out* para o baseado em SAT com Compactação. Os dados obtidos por intermédio da validação do método – e que servirão como base das análises seguintes – podem ser vistos no Apêndice B.9.

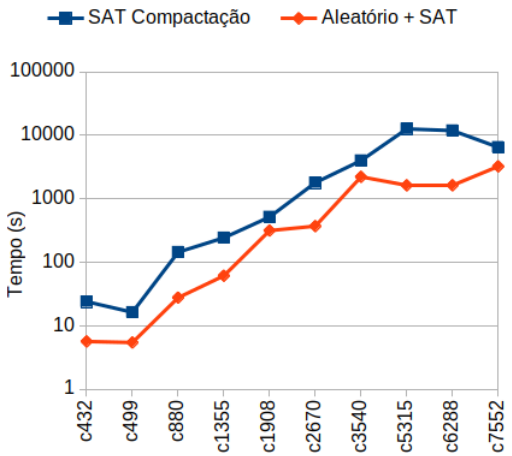
Para as análises realizadas nesta seção, com propósito de comparação, serão utilizados os dados obtidos utilizando parâmetros de limite e *Time out* igual à 4 e $0,01s$, respectivamente. Diferentemente do método baseado em SAT utilizado sozinho (no qual o seu melhor caso se encontrava com $1s$ de *Time out*), no método híbrido o *Time out* é reduzido e o método aleatório é utilizado um limite baixo. Essa escolha de parametrização é justificável, uma vez que ao reduzir o *Time out* do resolvedor SAT o método se torna mais rápido, porém apresenta uma perda na cobertura de falhas. Com isso, o método aleatório deve ser rápido e eficiente o suficiente para suprir a mencionada redução de desempenho.

A análise comparativa entre o método híbrido e a metodologia baseada em SAT com Compactação é ilustrada na Figura 38. Como o ponto negativo da metodologia baseada em SAT com Compactação é o aumento do tempo de execução em comparação à metodologia clássica, primeiro será analisado o tempo de execução apresentado na Figura 38a. Foi utilizado o mesmo tempo de *Time out* para as duas aplicações.

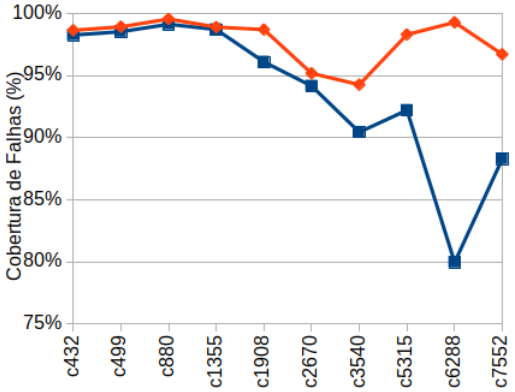
Nesse cenário, a metodologia híbrida resultou na redução do tempo de execução de 40% no pior caso e 87% no melhor. Levando em consideração que o método aleatório é mais rápido que o SAT com Compactação, ao executar o método aleatório antes, a carga de falhas para o método baseado em SAT é reduzida, fazendo com que o tempo final também seja reduzido. Esses valores podem sofrer leve variação devido à uma parcela das falhas serem processadas pelo método aleatório. No entanto, a redução do tempo de execução é alta o suficiente para que o comportamento se mantenha.

A próxima análise é feita comparando a cobertura de falhas, ilustrada no gráfico da Figura 38b. O método baseado em SAT com Compactação com $0,01s$ de *Time out* consegue atingir níveis acima de 95% de cobertura para os primeiros circuitos do conjunto de *benchmark*. Conforme a complexidade dos circuitos aumenta, o *Time out* baixo começa a não ser suficiente para alcançar bons níveis de cobertura de falhas. Ao utilizar o método híbrido, o método aleatório consegue contornar essa defasagem do método baseado em SAT, fazendo com que o método híbrido consiga atingir níveis mais altos de cobertura de falhas em um tempo menor.

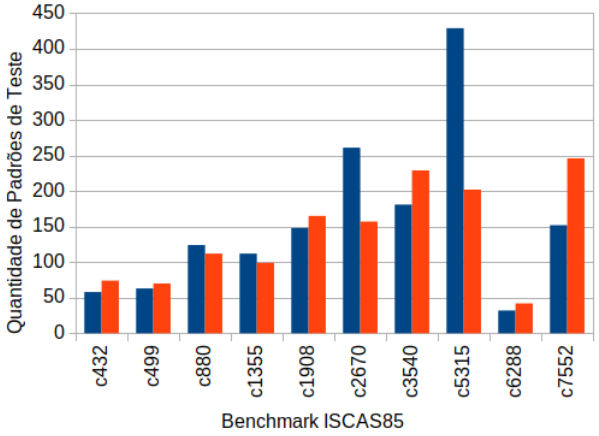
A análise restante é feita baseada na comparação do tamanho do conjunto de padrões de teste, visto no gráfico da Figura 38c. O SAT com Compactação utiliza-se de uma



(a) Comparação de Tempo de Execução.



(b) Comparação de Cobertura de Falhas.



(c) Comparação de Quantidade de Padrões de Teste.

Figura 38 – Comparação entre método híbrido Aleatório + SAT com Compactação e SAT com Compactação individual.

metodologia para redução dos padrões de teste, ao passo que o método aleatório depende do padrão gerado aleatoriamente. Como no modelo híbrido parte dos padrões gerados depende do método aleatório, em alguns casos, esse último pode encontrar soluções melhores. Isso resulta em conjuntos menores, sendo análogo para o caso inverso no qual pode refletir em conjuntos maiores. Deve-se verificar, então, se o ganho de desempenho no tempo de execução e na cobertura de falhas compensa a variação da quantidade de padrões de teste.

Para analisar a relação dos três fatores, foi realizada a comparação da Figura de Mérito do método híbrido com a implementação baseada em SAT com Compactação. A Figura 39 demonstra graficamente a comparação entre os métodos. Pode ser destacado que – em comparação ao método individual baseado em SAT com Compactação – o método híbrido apresentou resultados melhores. A redução significativa do tempo de execução e

uma cobertura de falhas maior fez com que a performance final do método híbrido aumentasse em relação à sua versão individual. Mesmo em casos em que houve um aumento mais significante nos padrões de teste, como se dá nos circuitos c3540 e c7552, o ganho de performance ainda compensou resultando em um valor da Figura de Mérito menor.

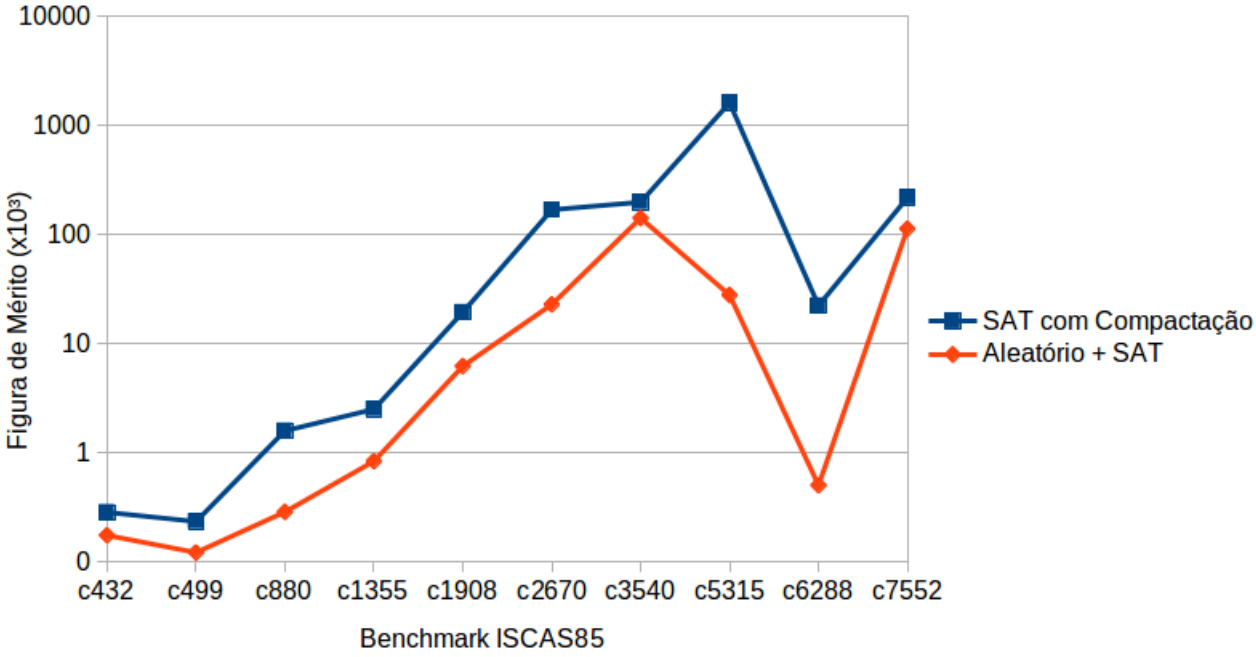


Figura 39 – Comparação da Figura de Mérito para SAT com Compactação individual e Aleatório + SAT com Compactação. $((1 - FC) \cdot \#P^2 \cdot TE^{\frac{1}{2}})$

5.6.3 Aleatório Estrutural, ATPG baseado em SAT com Compactação e Algoritmo D

A última implementação a ser abordada será o método híbrido composto pelos métodos Aleatório Estrutural, baseado em SAT com Compactação e Algoritmo-D. Nesta seção será abordada e analisada a interação entre três métodos ATPG diferentes: o Aleatório para redução de carga, o método baseado em SAT com facilidade para falhas difíceis e o Algoritmo-D para as falhas fáceis, executados nessa ordem, respectivamente. A execução do método baseado em SAT antes do Algoritmo-D possui dois aspectos interessantes: a desconsideração das falhas impossíveis de serem testadas identificadas pelo SAT e a redução de carga sobre o Algoritmo-D.

A metodologia dos parâmetros foi mantida: limite de parada igual a 4, *Time out* igual a $0,01s$ e *backtrack* igual a 1. Identificou-se que, adicionando-se o método aleatório ao baseado em SAT com Compactação, foi apresentado um acréscimo de performance. Essa aplicação híbrida segue a mesma lógica, será analisado o comportamento do método ao adicionar o Algoritmo-D ao final da execução do método híbrido. Utilizando-se os três métodos em conjunto, o esperado é que haja aproveitamento de uma melhor maneira nas

características de cada método. A validação do método corrente pode ser conferida no Apêndice B.10.

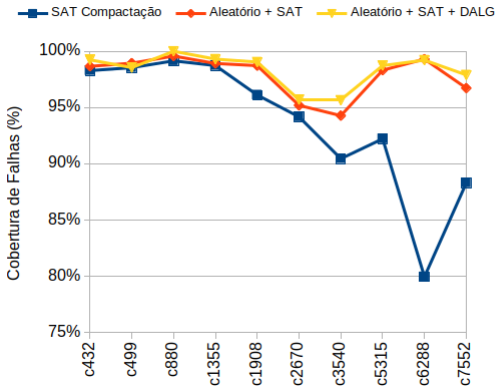
A análise comparativa da metodologia híbrida corrente pode ser vista na Figura 40. O primeiro fator a ser analisado é a cobertura de falhas, conforme se encontra na Figura 40a. Foi visto que ao utilizar o método baseado em SAT com Compactação, a cobertura de falhas quando usado *Time out* igual a $0,01s$ reduzia conforme o aumento de complexidade dos circuitos. Adicionando o método aleatório à execução, foi possível contornar o problema resultando em uma cobertura de falhas mais consistente. Seguindo a mesma lógica, ao adicionar o método Algoritmo-D após a execução do método baseado em SAT com Compactação é apresentada uma pequena progressão da taxa de cobertura de falhas.

No geral, a média de aumento da cobertura se aproximou de $0,5\%$, onde em casos como os circuitos c3540 e c7552 resultaram em um aumento de pouco mais de 1%, ao passo que casos como o c499 foi verificada uma redução de $0,4\%$ quando comparada à metodologia híbrida Aleatório Estrutural e SAT com Compactação. Ao adicionar o Algoritmo-D ao final da execução, parte das falhas abortadas pelo método SAT são processadas com sucesso, gerando padrões e – consequentemente – aumentando a cobertura de falhas.

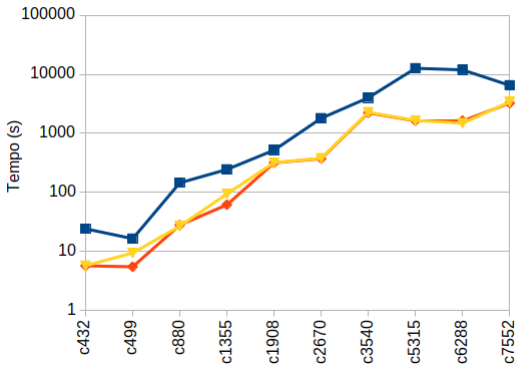
Contudo, o pequeno aumento é justificável por estar próximo de seu valor de saturação. Alguns casos de variação da cobertura de falhas podem ser também justificados pelo uso parcial da geração aleatória de padrões. Como é utilizado um valor baixo de limite de parada, sua aleatoriedade se torna mais alta, e, consequentemente, pequenas variações podem ocorrer. Tais variações podem se refletir também nos demais fatores, já que dependendo dos padrões gerados aleatoriamente, o conjunto de falhas a serem analisadas – tanto pelo SAT, quanto pelo Algoritmo-D – são diferentes.

A segunda análise é baseada no tempo de execução obtido pelas implementações. A Figura 40b demonstra o comportamento temporal das três aplicações em comparativo. Analisando o gráfico correspondente, pode-se concluir que mesmo adicionando o método Algoritmo-D à execução da aplicação, o tempo de execução se manteve no mesmo comportamento do método híbrido sem o Algoritmo-D. Porém, são apresentadas duas exceções: circuitos c499 e c1355. Nesses, verificou-se um aumento no tempo de execução ao se comparar com o método híbrido Aleatório e SAT com Compactação. O referido aumento é justificável pelas limitações do Algoritmo-D relacionadas a portas com lógicas mais complexas, onde o Algoritmo-D acaba por gastar maior tempo de execução para o processamento dessas falhas.

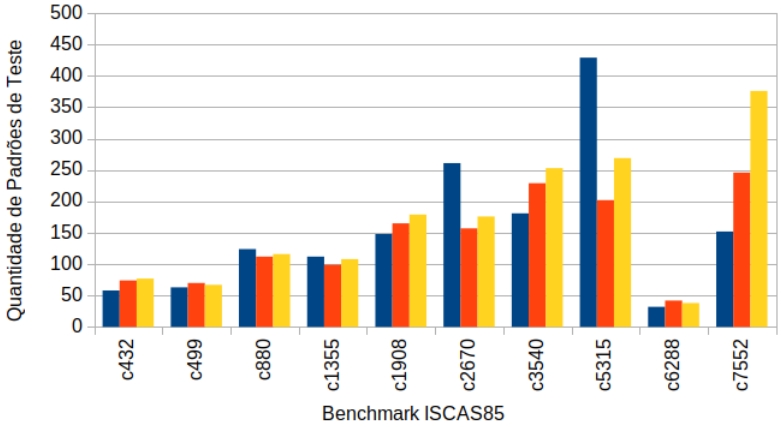
O terceiro fator a ser analisado é o tamanho do conjunto de padrões de teste gerado pelos métodos. No gráfico da Figura 40c, pode ser visto o comparativo das três aplicações. Em todos os cenários, a metodologia corrente apresenta um aumento no conjunto de padrões em comparação com às demais implementações. Tal acréscimo é justificado pelo



(a) Comparação de Cobertura de Falhas.



(b) Comparação de Tempo de Execução.



(c) Comparação de Quantidade de Padrões de Teste.

Figura 40 – Comparação entre método híbrido Aleatório + SAT + DALG, Aleatório + SAT e SAT com Compactação na versão individual.

aumento da cobertura de falhas proveniente da execução do Algoritmo-D. Em outros casos, a variação de padrões pode ser causada também pelo método aleatório, já que a execução dele também impacta na execução de métodos subsequentes, conforme dito anteriormente.

Para verificar a performance do método corrente são relacionados os três fatores analisados por meio do cálculo da Figura de Mérito. Com essa métrica proposta, é possível verificar se o pequeno ganho de cobertura de falhas – mantendo um tempo de execução semelhante – compensa o aumento do número de padrões de teste. A comparação da Figura de Mérito dos métodos em análise é apresentada no gráfico da Figura 41.

É visto que em seis dos dez circuitos do *benchmark*, o método híbrido em análise possui performance próxima ou maior que a metodologia Aleatório e SAT com Compactação. Nesses casos, o ganho na cobertura de falhas compensou o aumento do conjunto de teste. Entretanto, casos como os que se apresentam nos circuitos c499, c2670, c5315 e c7552 obtiveram uma performance menor, de acordo com a configuração feita para a

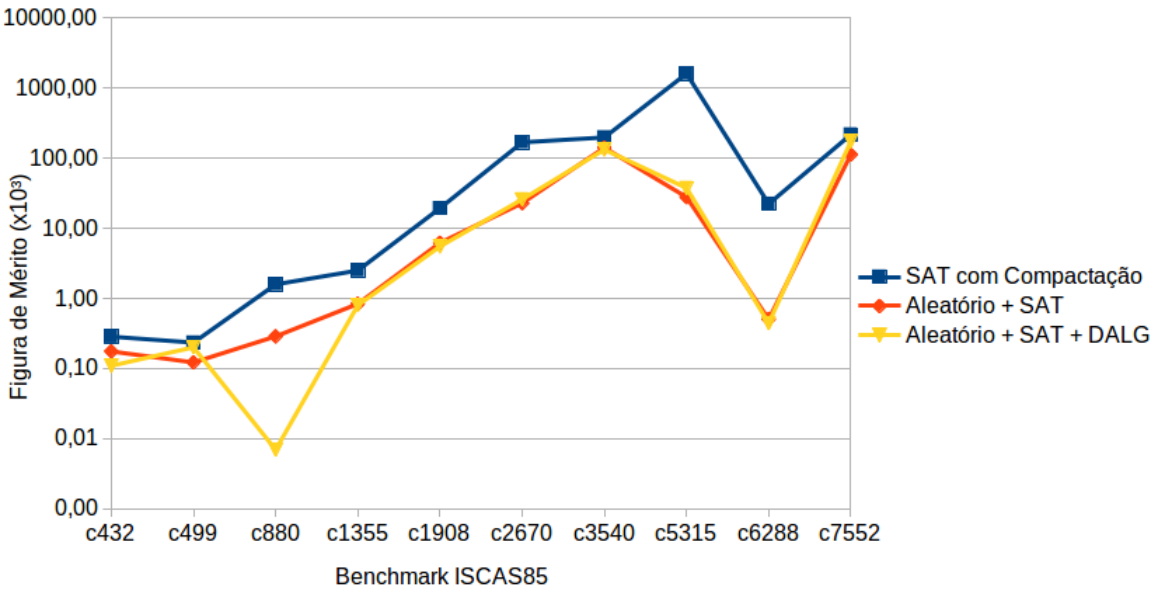


Figura 41 – Comparação da Figura de Mérito para Aleatório + SAT + DALG e demais implementações. $((1 - FC) \cdot \#P^2 \cdot TE^{\frac{1}{2}})$

Figura de Mérito.

Mesmo que o método corrente apresente um leve aumento ou redução de performance de acordo com o circuito, há certos pontos interessantes a serem analisados. Com o Algoritmo-D – mesmo em se tratando de um método lento – não houve grande variação no tempo de execução em relação à outra metodologia híbrida. Além disso, conseguiu um aumento na cobertura de falhas na maioria dos casos, ainda que pouco e mesmo que parte dele seja reflexo da variação do método aleatório. Pode-se concluir que, com esse comportamento verificado, leva-se a pensar que ao utilizar métodos estruturais mais avançados com maior performance, o cenário para metodologia contendo os três tipos de ATPG se torne a melhor opção em relação às demais.

6 Conclusões

Com o avanço tecnológico guiado pela Lei de Moore, a miniaturização e o aumento da densidade de circuitos integrados tornaram etapas – incluindo a de teste – cada vez mais complexas de serem executadas. Ao longo dos anos, diversos meios para a facilitação da etapa de teste foram propostos: desde projetar o circuito visando uma maior testabilidade, até modelar algoritmos para a geração de conjuntos reduzidos para teste. Com essa premissa, é de suma importância buscar o avanço de técnicas e heurísticas para se promover um processo de teste cada vez mais otimizado.

O trabalho em questão teve como foco a etapa de teste de circuitos integrados, especificamente a análise de algoritmos de geração automática de padrões de teste ou ATPGs. Atualmente, duas principais abordagens são utilizadas: estrutural e baseada em SAT. Metodologias estruturais visam a realização de uma análise topológica dos circuitos por meio de estruturas de dados como grafos, ao passo que abordagens baseadas em SAT realizam uma análise algébrica e funcional, modelando o circuito por intermédio de uma expressão Booleana conhecida como CNF.

Para um melhor entendimento, o trabalho explorou vasto referencial teórico, abordando conceitos, tais como a diferença entre defeito, falha, erro e falha sistêmica, bem como terminologias comumente utilizadas: modelagem de falhas, *fault collapsing*, etapa e geração de teste e seus algoritmos, dentre outros assuntos pertinentes.

De mesmo modo, também foi realizada uma investigação do estado-da-arte no que tange o assunto ATPG para compreender a pesquisa atual. Por meio das buscas conceituais realizadas, foi concluído que abordagens estruturais apresentam maior facilidade para gerar padrões voltados às falhas fáceis de testar, à medida que abordagens baseadas em SAT possuem facilidade para gerar padrões direcionadas às falhas difíceis de testar. Com esse entendimento, o trabalho teve como objetivo realizar uma análise sobre a eficiência desses métodos híbridos com o intuito de ressaltar e difundir a relevância de tal metodologia.

Para alcançar o objetivo do trabalho foi desenvolvida uma ferramenta ATPG contendo cinco métodos diferentes e três métodos híbridos estruturados. Dos cinco métodos, dois deles são aleatórios (sendo um com base estrutural e outro baseado em SAT); há também um método estrutural utilizando o Algoritmo-D e, por fim, outros dois métodos baseados em SAT (sendo uma implementação clássica e outro com abordagem de compactação de padrões). Os três métodos híbridos estruturados são: a) Aleatório Estrutural e Algoritmo-D; b) Aleatório Estrutural e SAT com Compactação; c) um último contendo os três, Aleatório Estrutural com SAT com Compactação e Algoritmo-D.

Os métodos estruturados foram validados e todos os dados gerados foram indexados no Apêndice B. Diversas análises foram realizadas tanto entre os métodos em suas versões individuais, como também em suas abordagens híbridas. Foram apresentadas as principais características de cada módulo: o Aleatório, que possui uma baixa complexidade e, consequentemente, performance para redução de carga; o Algoritmo-D, que apesar de ser um algoritmo clássico possui certa notoriedade na geração de padrões para falhas fáceis; os métodos baseados em SAT, que vieram como alternativa aos métodos estruturais com sua maior capacidade de saturação rápida de cobertura de falhas. Com suas características definidas, os módulos híbridos, bem como as respectivas análises puderam ser feitas.

A primeira análise de método híbrido se deu pelo Aleatório com o Algoritmo-D. Concluiu-se que com a adição do módulo aleatório em conjunto com o clássico Algoritmo-D, houve um aumento de performance e melhoria em todos os principais fatores (padrões de teste, cobertura de falhas, tempo de execução e a métrica FoM) ao se comparar à implementação individual do Algoritmo-D. O mesmo se deu ao se comparar ao SAT Clássico: o método híbrido obteve um melhor resultado na maioria dos casos.

A segunda análise de método híbrido foi feita usando Aleatório com SAT com Compactação. O método SAT com Compactação consegue obter um ótimo conjunto reduzido de padrões como também uma taxa de cobertura de falhas máxima na maioria dos casos, porém – para alcançar esse resultado – um alto tempo de execução era preciso. Ao combinar com um método aleatório, a cobertura de falhas se manteve, os padrões aumentaram sutilmente devido à metodologia aleatória em conjunto; entretanto, ao analisar a métrica FoM, a redução do tempo de execução compensou o aumento dos padrões, aumentando a performance do método.

A última análise híbrida feita foi utilizando o Aleatório, SAT com Compactação e Algoritmo-D. Comparou-se com o método híbrido mais eficiente até então: o Aleatório com SAT com Compactação. Concluiu-se que mesmo utilizando o Algoritmo-D como parte estrutural (um método pouco otimizado), a cobertura de falhas aumentou na maioria dos casos. Mesmo que parte desses seja um reflexo mais do método aleatório do que o Algoritmo-D, o tempo de execução se manteve.

Dentre os métodos híbridos apresentados, pode-se dizer que a combinação mais promissora é a que contém os três métodos em conjunto. Utilizando-os, o aproveitamento de suas características positivas é maior. Ainda que o Algoritmo-D apresente um desempenho baixo e que em alguns casos seja apenas o reflexo do método aleatório, o ganho de performance – embora pequeno – foi visto na maioria dos circuitos do *benchmark*.

O esperado é que ao utilizar um método estrutural com uma performance melhor, o ganho de desempenho será maior para esta abordagem. Outro aspecto relevante para um maior aumento de performance seriam meios de classificação de falhas, onde elas seriam

distribuídas entre os métodos de acordo com a sua dificuldade. Com isso, o aproveitamento de suas características seria melhor e a diferença na performance entre os métodos seria maior.

Fundamentando-se nas pesquisas, implementações e análises realizadas, pode-se concluir que o objetivo do trabalho foi atingido. A relevância da performance foi demonstrada quando feita a combinação de diferentes métodos ATPGs em conjunto. Assim, ao se estruturarem métodos ATPGs em conjunto de forma coesa e com uma configuração de parâmetros lógica de acordo com as características de cada método, foi possível alcançar um aumento de desempenho em comparação a métodos que utilizam uma só abordagem, reconhecendo a necessidade de se conhecer e buscar a exploração das características de cada método ATPG com a finalidade de uma interação para se obter cada vez mais alto de performance.

Referências

AGRAWAL, V. D.; PRASAD, A. V. S. S.; ATRE, M. V. Fault collapsing via functional dominance. In: *Test Conference, 2003. Proceedings. ITC 2003. International*. [S.l.: s.n.], 2003. v. 1, p. 274–280. ISSN 1089-3539. Citado 3 vezes nas páginas 25, 26 e 62.

ALI, L. G.; HUSSEIN, A. I.; ALI, H. M. Parallelization of unit propagation algorithm for sat-based atpg of digital circuits. In: *2016 28th International Conference on Microelectronics (ICM)*. [S.l.: s.n.], 2016. p. 184–188. Citado na página 44.

BRGLEZ, F.; BRYAN, D.; KOZMINSKI, K. Combinational profiles of sequential benchmark circuits. In: *IEEE International Symposium on Circuits and Systems,*. [S.l.: s.n.], 1989. p. 1929–1934 vol.3. Citado na página 44.

BRYAN, D. The iscas'85 benchmark circuits and netlist format. *North Carolina State University*, v. 25, 1985. Citado 9 vezes nas páginas 7, 9, 22, 31, 32, 44, 62, 63 e 94.

BURCHARD, J. et al. Evaluating the effectiveness of d-chains in sat-based atpg. In: *2017 18th IEEE Latin American Test Symposium (LATS)*. [S.l.: s.n.], 2017. p. 1–6. Citado 2 vezes nas páginas 37 e 43.

BUSHNELL, M. L.; AGRAWAL, V. D. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. [S.l.]: Kluwer Academic Publishers, 2000. Citado na página 21.

CHEN, H.; MARQUES-SILVA, J. Tg-pro: A new model for sat-based atpg. In: *2009 IEEE International High Level Design Validation and Test Workshop*. [S.l.: s.n.], 2009. p. 76–81. ISSN 1552-6674. Citado na página 44.

CHEN, H.; MARQUES-SILVA, J. A two-variable model for sat-based atpg. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 32, n. 12, p. 1943–1956, Dezembro 2013. ISSN 0278-0070. Citado 5 vezes nas páginas 7, 18, 39, 40 e 44.

CORNO, F.; REORDA, M. S.; SQUILLERO, G. Rt-level itc'99 benchmarks and first atpg results. *IEEE Design Test of Computers*, v. 17, n. 3, p. 44–53, July 2000. ISSN 0740-7475. Citado na página 44.

DRECHSLER, R. et al. On acceleration of sat-based atpg for industrial designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 27, n. 7, p. 1329–1333, July 2008. ISSN 0278-0070. Citado 2 vezes nas páginas 18 e 44.

EGGERSGLÜSS, S. et al. On optimization-based atpg and its application for highly compacted test sets. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, IEEE, v. 35, n. 12, p. 2104–2117, 2016. Citado na página 45.

EéN, N.; SöRENSSON, N. An extensible sat solver. *Proc. International Conference of Theory and Applications of Satisfiability Testing*, p. 502–518, 2004. Citado 3 vezes nas páginas 18, 43 e 53.

FUJIWARA, H.; SHIMONO, T. On the acceleration of test generation algorithms. *IEEE Transactions on Computers*, C-32, n. 12, p. 1137–1144, Dezembro 1983. ISSN 0018-9340. Citado 5 vezes nas páginas 17, 29, 31, 44 e 68.

GIZDARSKI, E.; FUJIWARA, H. Spirit: a highly robust combinational test generation algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 21, n. 12, p. 1446–1458, Dec 2002. ISSN 0278-0070. Citado na página 29.

GOEL, P. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Transactions on Computers*, C-30, n. 3, p. 215–222, Março 1981. ISSN 0018-9340. Citado 3 vezes nas páginas 17, 29 e 68.

HAMZAOGLU, I.; PATEL, J. H. New techniques for deterministic test pattern generation. In: *Proceedings. 16th IEEE VLSI Test Symposium (Cat. No.98TB100231)*. [S.l.: s.n.], 1998. p. 446–452. ISSN 1093-0167. Citado na página 29.

LARRABEE, T. Test pattern generation using boolean satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 11, n. 1, p. 4–15, Janeiro 1992. ISSN 0278-0070. Citado 9 vezes nas páginas 18, 29, 37, 38, 39, 41, 43, 54 e 55.

LEE, H.; HA, D. Atalanta: an efficient atpg for combinational circuits. In: . [S.l.: s.n.], 1993. Citado na página 44.

LIOY, A. Looking for functional fault equivalence. In: *Test Conference, 1991, Proceedings., International*. [S.l.: s.n.], 1991. p. 858–. ISSN 1089-3539. Citado na página 26.

MARQUES-SILVA, J. P.; SAKALLAH, K. A. Grasp: a search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, v. 48, n. 5, p. 506–521, Maio 1999. ISSN 0018-9340. Citado 2 vezes nas páginas 18 e 43.

MATSUNAGA, Y. An accelerating technique for sat-based atpg. In: *IPSJ Transactions on System LSI Design Methodology*. [S.l.: s.n.], 2017. p. 39–44. Citado 2 vezes nas páginas 43 e 44.

MOSKEWICZ, M. W. et al. Chaff: engineering an efficient sat solver. In: *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*. [S.l.: s.n.], 2001. p. 530–535. ISSN 0738-100X. Citado 2 vezes nas páginas 18 e 43.

NAVABI, Z. *Digital system test and testable design*. [S.l.]: Springer, 2011. Citado 5 vezes nas páginas 9, 22, 23, 24 e 30.

PINTO, A. C. M.; REIS, R. *Um Estudo De Técnicas De Aceleração Para Algoritmos De Análise De Timing Funcional Baseados Em Geração Automática De Teste*. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul, 2002. Citado na página 30.

ROTH, J. P. Diagnosis of automata failures: A calculus and a method. *IBM Journal of Research and Development*, v. 10, n. 4, p. 278–291, Julho 1966. ISSN 0018-8646. Citado 7 vezes nas páginas 7, 17, 29, 32, 36, 53 e 66.

SANDIREDDY, R. K. K. R.; AGRAWAL, V. D. Diagnostic and detection fault collapsing for multiple output circuits. In: *Design, Automation and Test in Europe*. [S.l.: s.n.], 2005. p. 1014–1019 Vol. 2. ISSN 1530-1591. Citado 2 vezes nas páginas 7 e 26.

SCHULZ, M. H.; TRISCHLER, E.; SARFERT, T. M. Socrates: a highly efficient automatic test pattern generation system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 7, n. 1, p. 126–137, Jan 1988. ISSN 0278-0070. Citado na página 29.

TSEITIN, G. S. On the complexity of derivation in propositional calculus. In: _____. *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983. p. 466–483. ISBN 978-3-642-81955-1. Disponível em: <https://doi.org/10.1007/978-3-642-81955-1_28>. Citado na página 38.

UBAR, R. et al. Structural fault collapsing by superposition of bdds for test generation in digital circuits. In: *Quality Electronic Design (ISQED), 2010 11th International Symposium on*. [S.l.: s.n.], 2010. p. 250–257. ISSN 1948-3287. Citado 2 vezes nas páginas 24 e 62.

Université d'Artois. *SAT4J, the boolean satisfaction and optimization library in Java*. 2017. Disponível em: <http://www.sat4j.org/>. Citado na página 53.

WANG, L.; CHANG, Y.; CHENG, T. *VLSI Test Principles and Architectures*. [S.l.]: Morgan Kaufmann Publishers, 2006. Citado 5 vezes nas páginas 7, 19, 24, 28 e 29.

WANG, L.; WU, C.; WEN, X. *Electronic Design Automation*. [S.l.]: Morgan Kaufmann Publishers, 2009. Citado 2 vezes nas páginas 27 e 57.

WESTE, N. H.; HARRIS, D. *CMOS VLSI design: a circuits and systems perspective*. [S.l.]: Pearson Education, 2011. Citado 3 vezes nas páginas 7, 27 e 28.

Apêndices

APÊNDICE A – Conjunto *Benchmark* ISCAS85

Tabela 10 – Conjunto de *Benchmark* ISCAS85 (BRYAN, 1985)

Circuitos	Função do Circuito	#Portas	#Sinais	#PI	#PO	#Falhas	#Falhas FC
c17	-	6	11	5	2	34	22
c432	Decodificar de Prioridade	160	196	36	7	864	524
c499	ECAT	202	243	41	32	998	758
c880	ULA e Controle	383	443	60	26	1760	942
c1355	ECAT	546	587	41	32	2710	1574
c1908	ECAT	880	913	33	25	3816	1879
c2670	ULA e Controle	1193	1350	233	140	5340	2747
c3540	ULA e Controle	1669	1719	50	22	7080	3428
c5315	ULA e Controle	2307	2485	178	123	10630	5350
c6288	Multiplicador de 16-bits	2416	2448	32	32	12576	7744
c7552	ULA e Controle	3512	3718	207	108	15104	7550

APÊNDICE B – Validação dos Métodos Implementados

B.1 Método Aleatório Baseado em SAT

Tabela 11 – Validação do Método Aleatório baseado em SAT usando Limite = 1 e 2

	Limite = 1				Limite = 2			
Circuitos	#Padrões	#Abortadas	Cobertura	Tempo (s)	#Padrões	#Abortadas	Cobertura	Tempo (s)
c17	5	4	81,82%	0,37	6	1	95,45%	0,35
c432	21	169	67,75%	22,51	38	71	86,45%	28,39
c499	10	175	76,91%	11,79	8	230	69,66%	16,47
c880	18	217	76,96%	31,76	39	157	83,33%	60,47
c1355	7	744	52,73%	137,70	13	522	66,84%	220,55
c1908	19	727	61,31%	340,10	17	778	58,60%	302,07
c2670	47	724	73,64%	571,52	47	676	75,39%	535,16
c3540	33	966	71,82%	2.208,20	46	944	72,46%	2.868,13
c5315	43	635	88,13%	1.976,82	56	505	90,56%	1.707,39
c6288	25	205	97,35%	7.065,66	46	56	99,28%	7.028,06
c7552	44	1.523	79,83%	3.633,99	60	1.268	83,21%	4.379,47

Tabela 12 – Validação do Método Aleatório baseado em SAT usando Limite = 4 e 8

	Limite = 4				Limite = 8			
Circuitos	#Padrões	#Abortadas	Cobertura	Tempo (s)	#Padrões	#Abortadas	Cobertura	Tempo (s)
c17	9	0	100,00%	0,20	8	0	100,00%	0,25
c432	40	66	87,40%	30,70	61	31	94,08%	38,12
c499	38	66	91,29%	29,36	31	77	89,84%	23,78
c880	55	93	90,13%	75,41	69	87	90,76%	96,41
c1355	40	255	83,80%	373,75	56	189	87,99%	424,51
c1908	29	446	76,26%	335,10	41	468	75,09%	715,67
c2670	59	589	78,56%	666,21	76	528	80,78%	951,49
c3540	63	953	72,20%	3.819,06	119	454	86,76%	5.966,37
c5315	90	369	93,10%	2.305,72	142	196	96,34%	2.912,15
c6288	46	50	99,35%	7.982,88	45	54	99,30%	8.790,34
c7552	137	854	88,69%	7.158,94	177	731	90,32%	10.306,47

Tabela 13 – Validação do Método Aleatório baseado em SAT usando Limite = 16 e 32

	Limite = 16				Limite = 32			
Circuitos	#Padrões	#Abortadas	Cobertura	Tempo (s)	#Padrões	#Abortadas	Cobertura	Tempo (s)
c17	7	0	100,00%	0,21	6	0	100,00%	0,19
c432	62	43	91,79%	53,71	74	15	97,14%	47,40
c499	35	64	91,56%	39,02	50	24	96,83%	34,83
c880	80	46	95,12%	106,22	87	39	95,86%	167,68
c1355	74	92	94,16%	415,65	74	105	93,33%	557,52
c1908	41	550	70,73%	804,50	93	242	87,12%	1.377,71
c2670	70	582	78,81%	920,25	89	483	82,42%	1.694,13
c3540	177	241	92,97%	7.478,50	174	282	91,77%	8.282,70
c5315	157	142	97,35%	3.030,47	185	100	98,13%	3.911,55
c6288	52	36	99,54%	7.901,22	63	34	99,56%	9.294,40
c7552	177	701	90,72%	11.374,48	205	677	91,03%	14.990,85

B.2 Método Aleatório Estrutural

Tabela 14 – Validação do Método Aleatório Estrutural usando Limite = 1 e 2

	Limite = 1				Limite = 2			
Circuitos	#Padrões	#Abortadas	Cobertura	Tempo (s)	#Padrões	#Abortadas	Cobertura	Tempo (s)
c17	4	2	90,91%	0,3006	6	0	100,00%	0,2104
c432	6	328	37,40%	0,7526	13	239	54,39%	1,1337
c499	8	188	75,20%	0,9075	11	188	75,20%	1,0182
c880	24	214	77,28%	4,4853	52	134	85,77%	8,8666
c1355	7	769	51,14%	16,56	11	639	59,40%	17,8007
c1908	13	939	50,03%	40,1656	33	827	55,99%	93,4087
c2670	34	888	67,67%	66,1003	56	741	73,03%	101,6226
c3540	41	1343	60,82%	475,3864	62	1144	66,63%	665,3976
c5315	40	1354	74,69%	451,7757	67	707	86,79%	709,6696
c6288	23	154	98,01%	1083,7691	36	62	99,20%	1501,5935
c7552	55	1597	78,85%	1181,1385	124	1092	85,54%	2404,4596

Tabela 15 – Validação do Método Aleatório Estrutural usando Limite = 4 e 8

	Limite = 4				Limite = 8			
Circuitos	#Padrões	#Abortadas	Cobertura	Tempo (s)	#Padrões	#Abortadas	Cobertura	Tempo (s)
c17	6	1	95,45%	0,3783	5	0	100,00%	0,242
c432	50	74	85,88%	2,8299	54	51	90,27%	3,2437
c499	17	111	85,36%	1,1816	29	63	91,69%	2,16
c880	63	102	89,17%	11,0118	71	73	92,25%	11,4918
c1355	39	217	86,21%	30,0878	41	187	88,12%	29,6629
c1908	36	637	66,10%	92,7864	43	540	71,26%	112,6234
c2670	54	721	73,75%	107,6392	66	663	75,86%	141,5557
c3540	97	871	74,59%	954,253	112	830	75,79%	1133,5282
c5315	109	346	93,53%	1082,6061	136	288	94,62%	1599,2375
c6288	40	59	99,24%	1666,3248	44	42	99,46%	2044,8407
c7552	149	998	86,78%	3019,7063	194	865	88,54%	4455,1584

Tabela 16 – Validação do Método Aleatório Estrutural usando Limite = 16 e 32

	Limite = 16				Limite = 32			
Circuitos	#Padrões	#Abortadas	Cobertura	Tempo (s)	#Padrões	#Abortadas	Cobertura	Tempo (s)
c17	5	0	100,00%	0,2575	7	0	100,00%	0,2556
c432	75	23	95,61%	6,3277	77	14	97,33%	10,0213
c499	53	33	95,65%	4,8355	55	16	97,89%	7,682
c880	76	82	91,30%	19,6431	91	50	94,69%	39,343
c1355	72	167	89,39%	65,4837	79	124	92,12%	89,1619
c1908	141	216	88,50%	620,9838	156	123	93,45%	689,0964
c2670	87	634	76,92%	240,7874	104	529	80,74%	416,9022
c3540	195	508	85,18%	2204,9849	245	356	89,61%	3629,1771
c5315	184	162	96,97%	2779,517	199	135	97,48%	4630,006
c6288	50	34	99,56%	3015,3146	52	34	99,56%	4215,1446
c7552	238	731	90,32%	7452,9436	241	762	89,91%	10038,1397

B.3 Método ATPG Estrutural: Algoritmo-D

Tabela 17 – Validação do Método ATPG Estrutural Algoritmo-D (*Backtrack=1*) e Cálculo da Métrica FoM: $(1 - FC) \cdot \#P^2 \cdot TE^{\frac{1}{2}}$

Circuitos	#Padrões	#Abortadas	Cobertura	Tempo (s)	FoM
c17	13	0	100,00%	0,48	0,01
c432	200	24	95,42%	179,48	24.598,13
c499	76	486	35,88%	241,06	57.507,01
c880	463	0	100,00%	978,03	670,40
c1355	50	1300	17,41%	6.993,63	172.696,03
c1908	375	818	56,47%	25.859,18	9.846.826,30
c2670	928	536	80,49%	71.001,22	44.797.894,90
c3540	1076	913	73,37%	114.437,75	104.352.320,59
c5315	2031	595	88,88%	494.336,12	322.838.045,46

B.4 Método ATPG baseado em SAT Clássico sem uso de *Fault Collapsing*

Tabela 18 – Validação do Método ATPG baseado em SAT Clássico sem uso de *Fault Collapsing* usando *Time Out* = 0,001s e 0,01s

	0,001s					0,01s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	11	0	0	100,00%	0,17	11	0	0	100,00%	0,14
c432	529	19	10	96,64%	4,41	530	14	10	97,22%	4,33
c499	405	41	8	95,09%	5,47	413	38	5	95,69%	5,65
c880	546	33	0	98,13%	11,20	542	33	0	98,13%	11,08
c1355	308	2216	0	18,23%	49,10	1190	72	8	97,05%	51,84
c1908	1279	525	9	86,01%	87,55	1294	510	9	86,40%	88,40
c2670	2023	1150	167	75,34%	144,45	2035	1071	165	76,85%	142,77
c3540	2939	2176	200	66,44%	389,77	2922	2204	189	66,20%	390,76
c5315	3282	3787	50	63,90%	601,81	3346	3608	48	65,61%	597,31
c6288	648	11200	4	10,91%	2.625,28	683	11180	4	11,07%	2.653,86
c7552	1713	12773	159	14,38%	1.638,40	2088	12247	178	17,74%	1.849,62

Tabela 19 – Validação do Método ATPG baseado em SAT Clássico sem uso de *Fault Collapsing* usando *Time Out* = 0,1s e 1s

	0,1s					1s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	11	0	0	100,00%	0,20	11	0	0	100,00%	0,15
c432	541	0	10	98,84%	4,45	541	0	10	98,84%	4,27
c499	435	0	8	99,20%	5,50	435	0	8	99,20%	5,50
c880	565	0	0	100,00%	11,72	565	0	0	100,00%	11,45
c1355	1240	0	8	99,70%	53,57	1240	0	8	99,70%	53,04
c1908	1577	3	9	99,69%	89,36	1580	0	9	99,76%	90,75
c2670	2306	0	190	96,44%	147,52	2306	0	190	96,44%	148,40
c3540	4056	21	242	96,29%	354,05	4067	0	250	96,47%	364,94
c5315	4566	8	62	99,34%	520,52	4573	0	62	99,42%	518,58
c6288	7093	2375	5	81,08%	3.127,54	8910	77	15	99,27%	3.210,04
c7552	7933	62	218	98,15%	1.336,88	7983	0	219	98,55%	1.356,69

Tabela 20 – Validação do Método ATPG baseado em SAT Clássico sem uso de *Fault Collapsing* usando *Time Out* = 10s e 100s

	10s					100s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	11	0	0	100,00%	0,15	11	0	0	100,00%	0,17
c432	541	0	10	98,84%	4,24	541	0	10	98,84%	4,27
c499	435	0	8	99,20%	5,36	435	0	8	99,20%	5,35
c880	565	0	0	100,00%	11,58	565	0	0	100,00%	11,25
c1355	1240	0	8	99,70%	53,11	1240	0	8	99,70%	54,26
c1908	1580	0	9	99,76%	90,78	1580	0	9	99,76%	90,18
c2670	2306	0	190	96,44%	148,83	2306	0	190	96,44%	147,71
c3540	4067	0	250	96,47%	371,23	4067	0	250	96,47%	362,13
c5315	4573	0	62	99,42%	522,69	4573	0	62	99,42%	519,40
c6288	8928	48	23	99,44%	3.712,03	8931	39	29	99,46%	7.505,90
c7552	7983	0	219	98,55%	1.357,08	7983	0	219	98,55%	1.360,54

B.5 Método ATPG baseado em SAT Clássico

Tabela 21 – Validação do Método ATPG baseado em SAT Clássico usando *Time Out* = 0,001s e 0,01s

	0,001s					0,01s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	11	0	0	100,00%	0,21	11	0	0	100,00%	0,23
c432	459	8	4	97,71%	2,87	450	16	4	96,18%	2,90
c499	381	42	8	93,40%	4,60	408	14	8	97,10%	4,61
c880	515	17	0	98,20%	6,59	515	20	0	97,88%	6,93
c1355	285	1211	0	23,06%	31,12	1020	39	8	97,01%	33,17
c1908	1020	247	7	86,48%	47,13	999	274	7	85,05%	43,47
c2670	1562	648	104	72,62%	73,53	1528	599	108	74,26%	75,25
c3540	1940	1037	87	67,21%	179,09	1914	1065	88	66,37%	179,50
c5315	2347	2220	44	57,68%	245,55	2477	1959	43	62,58%	265,92
c6288	510	6868	2	11,29%	1.544,30	491	6862	2	11,36%	1.548,19
c7552	1089	6237	100	16,07%	589,75	1026	6316	96	15,07%	584,58

Tabela 22 – Validação do Método ATPG baseado em SAT Clássico usando *Time Out* = 0,1s e 1s

	0,1s					1s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	11	0	0	100,00%	0,20	11	0	0	100,00%	0,14
c432	466	0	4	99,24%	2,23	466	0	4	99,24%	2,32
c499	418	0	8	98,94%	3,63	418	0	8	98,94%	4,07
c880	530	0	0	100,00%	6,07	530	0	0	100,00%	5,97
c1355	1056	0	8	99,49%	29,55	1056	0	8	99,49%	28,49
c1908	1216	0	7	99,63%	40,67	1216	0	7	99,63%	40,95
c2670	1871	0	115	95,81%	75,20	1871	0	115	95,81%	76,81
c3540	2790	10	127	96,00%	160,93	2796	0	131	96,18%	164,96
c5315	3872	1	59	98,88%	259,98	3873	0	59	98,90%	263,74
c6288	5961	999	3	87,06%	1.680,63	6907	44	7	99,34%	1.802,80
c7552	5967	8	130	98,17%	617,49	5974	0	131	98,26%	623,99

Tabela 23 – Validação do Método ATPG baseado em SAT Clássico usando *Time Out* = 10s e 100s

	10s					100s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	11	0	0	100,00%	0,11	11	0	0	100,00%	0,11
c432	466	0	4	99,24%	2,32	466	0	4	99,24%	2,32
c499	418	0	8	98,94%	3,81	418	0	8	98,94%	3,84
c880	530	0	0	100,00%	6,09	530	0	0	100,00%	5,92
c1355	1056	0	8	99,49%	28,82	1056	0	8	99,49%	28,65
c1908	1216	0	7	99,63%	40,86	1216	0	7	99,63%	41,48
c2670	1871	0	115	95,81%	77,21	1871	0	115	95,81%	76,94
c3540	2796	0	131	96,18%	162,61	2796	0	131	96,18%	163,46
c5315	3873	0	59	98,90%	263,55	3873	0	59	98,90%	263,28
c6288	6921	26	11	99,52%	2.059,70	6924	21	13	99,56%	4.034,85
c7552	5974	0	131	98,26%	617,46	5974	0	131	98,26%	624,65

Tabela 24 – Cálculo da Figura de Mérito com a Configuração: $(1 - FC) \cdot \#P^2 \cdot TE^{\frac{1}{2}}$

SAT Clássico	Figura de Mérito (FoM)					
Circuitos	0,001s	0,01s	0,1s	1s	10s	100s
c432	8.204,57	13.186,52	2.506,32	2.547,86	2.555,85	2.557,78
c499	20.573,13	10.412,69	3.547,38	3.770,32	3.634,67	3.649,84
c880	12.356,02	14.894,31	69,23	68,62	69,34	68,34
c1355	348.654,03	179.522,47	31.416,38	30.950,43	31.029,83	30.937,92
c1908	966.236,88	984.739,12	36.068,02	35.954,55	36.151,07	36.425,49
c2670	5.729.459,01	5.214.617,58	1.273.928,97	1.288.564,11	1.290.773,32	1.288.526,90
c3540	16.519.396,32	16.513.205,40	3.956.340,65	3.845.608,46	3.819.559,83	3.829.574,35
c5315	36.536.160,47	37.450.051,98	2.735.259,70	2.703.972,49	2.709.851,14	2.708.445,22
c6288	9.068.740,96	8.408.831,33	188.629.791,56	13.571.487,57	10.604.290,23	13.673.320,20
c7552	24.175.659,07	21.617.936,14	16.260.204,02	15.601.170,95	15.475.927,32	15.565.701,94

B.6 Método ATPG baseado em SAT com Compactação sem uso de *Fault Collapsing*

Tabela 25 – Validação do Método ATPG baseado em SAT com Compactação sem uso de *Fault Collapsing* usando *Time Out* = 0,001s e 0,01s

	0,001s					0,01s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	8	0	0	100,00%	0,27	8	0	0	100,00%	0,20
c432	54	1	10	98,73%	33,36	54	1	10	98,73%	33,88
c499	69	3	7	99,00%	38,01	72	3	8	98,90%	37,34
c880	124	15	0	99,15%	213,22	117	19	0	98,92%	208,56
c1355	73	453	0	83,28%	529,52	114	1	8	99,67%	406,42
c1908	152	60	8	98,22%	1.167,11	151	47	8	98,56%	1.035,98
c2670	293	84	177	95,11%	3.120,00	289	75	168	95,45%	2.827,63
c3540	189	281	208	93,09%	8.534,23	189	292	202	93,02%	8.270,63
c5315	522	492	49	94,91%	20.263,90	525	509	54	94,70%	21.572,36
c6288	33	1782	4	85,80%	19.374,23	34	1853	4	85,23%	21.539,24
c7552	174	1260	174	90,51%	12.234,76	177	1248	175	90,58%	12.745,90

Tabela 26 – Validação do Método ATPG baseado em SAT com Compactação sem uso de *Fault Collapsing* usando *Time Out* = 0,1s e 1s

	0,1s					1s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	8	0	0	100,00%	0,23	8	0	0	100,00%	0,24
c432	55	0	10	98,84%	34,20	55	0	10	98,84%	34,08
c499	69	0	8	99,20%	36,48	69	0	8	99,20%	36,55
c880	113	0	0	100,00%	164,30	113	0	0	100,00%	169,19
c1355	107	0	8	99,70%	370,35	107	0	8	99,70%	363,33
c1908	142	0	9	99,76%	751,83	142	0	9	99,76%	746,75
c2670	308	0	190	96,44%	2.870,55	308	0	190	96,44%	2.829,99
c3540	219	11	239	96,47%	7.786,40	219	0	250	96,47%	7.780,07
c5315	521	0	62	99,42%	20.222,27	521	0	62	99,42%	20.541,71
c6288	46	76	5	99,36%	10.948,78	41	54	15	99,45%	10.765,56
c7552	316	2	218	98,54%	22.762,36	316	0	219	98,55%	24.149,65

Tabela 27 – Validação do Método ATPG baseado em SAT com Compactação sem uso de *Fault Collapsing* usando *Time Out* = 10s e 100s

	10s					100s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	8	0	0	100,00%	0,31	8	0	0	100,00%	0,22
c432	55	0	10	98,84%	32,20	55	0	10	98,84%	29,75
c499	69	0	8	99,20%	35,63	69	0	8	99,20%	32,75
c880	113	0	0	100,00%	149,24	113	0	0	100,00%	147,97
c1355	107	0	8	99,70%	359,87	107	0	8	99,70%	363,72
c1908	142	0	9	99,76%	756,00	142	0	9	99,76%	725,93
c2670	308	0	190	96,44%	2.840,36	308	0	190	96,44%	2.813,26
c3540	219	0	250	96,47%	7.744,27	219	0	250	96,47%	7.710,95
c5315	521	0	62	99,42%	18.914,83	521	0	62	99,42%	19.555,87
c6288	41	45	23	99,46%	11.279,40	41	41	27	99,46%	14.667,90
c7552	316	0	219	98,55%	22.602,69	316	0	219	98,55%	23.052,79

B.7 Método ATPG baseado em SAT com Compactação

Tabela 28 – Validação do Método ATPG baseado em SAT com Compactação usando *Time Out* = 0,001s e 0,01s

	0,001s					0,01s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	7	0	0	100,00%	0,29	7	0	0	100,00%	0,27
c432	56	5	4	98,28%	24,87	58	5	4	98,28%	24,07
c499	61	3	8	98,55%	17,62	63	5	6	98,55%	16,40
c880	123	7	0	99,26%	131,01	124	8	0	99,15%	144,96
c1355	107	17	8	98,41%	204,66	112	14	6	98,73%	244,01
c1908	148	50	7	96,97%	493,18	148	66	7	96,12%	519,81
c2670	270	52	103	94,36%	1.894,87	261	57	103	94,18%	1.800,50
c3540	183	258	89	89,88%	3.996,32	181	236	91	90,46%	4.008,35
c5315	427	461	42	90,60%	13.342,27	429	372	44	92,22%	12.622,51
c6288	39	1838	2	76,24%	18.287,70	32	1549	2	79,97%	11.869,89
c7552	86	1420	99	79,88%	4.416,43	152	782	101	88,30%	6.494,61

Tabela 29 – Validação do Método ATPG baseado em SAT com Compactação usando *Time Out* = 0,1s e 1s

	0,1s					1s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	7	0	0	100,00%	0,30	7	0	0	100,00%	0,21
c432	59	0	4	99,24%	23,33	59	0	4	99,24%	22,61
c499	63	0	8	98,94%	16,63	63	0	8	98,94%	15,77
c880	123	0	0	100,00%	126,72	123	0	0	100,00%	124,97
c1355	111	0	8	99,49%	241,33	111	0	8	99,49%	232,42
c1908	147	0	7	99,63%	425,01	147	0	7	99,63%	442,64
c2670	289	0	115	95,81%	2.004,25	289	0	115	95,81%	1.943,94
c3540	203	8	124	96,15%	4.225,86	204	0	131	96,18%	4.084,95
c5315	464	0	59	98,90%	11.876,07	464	0	59	98,90%	11.656,80
c6288	42	40	2	99,46%	7.108,32	46	28	7	99,55%	6.827,32
c7552	281	3	130	98,24%	12.164,39	279	0	131	98,26%	11.325,20

Tabela 30 – Validação do Método ATPG baseado em SAT com Compactação usando *Time Out* = 10s e 100s

	10s					100s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	7	0	0	100,00%	0,21	7	0	0	100,00%	0,18
c432	59	0	4	99,24%	22,92	59	0	4	99,24%	22,82
c499	63	0	8	98,94%	16,31	63	0	8	98,94%	16,26
c880	123	0	0	100,00%	127,09	123	0	0	100,00%	122,32
c1355	111	0	8	99,49%	240,90	111	0	8	99,49%	231,00
c1908	147	0	7	99,63%	431,84	147	0	7	99,63%	436,59
c2670	289	0	115	95,81%	1.995,00	289	0	115	95,81%	2.032,53
c3540	204	0	131	96,18%	4.217,08	204	0	131	96,18%	4.216,96
c5315	464	0	59	98,90%	11.649,82	464	0	59	98,90%	11.598,35
c6288	44	23	11	99,56%	6.981,45	44	21	13	99,56%	9.032,03
c7552	279	0	131	98,26%	11.300,20	279	0	131	98,26%	11.343,32

Tabela 31 – Cálculo da Figura de Mérito com a Configuração: $(1 - FC) \cdot \#P^2 \cdot TE^{\frac{1}{2}}$

SAT Compactação	Figura de Mérito (FoM)					
Circuitos	0,001s	0,01s	0,1s	1s	10s	100s
c432	270,20	285,11	130,04	128,02	128,89	128,60
c499	228,23	234,85	172,46	167,92	170,77	170,53
c880	1.304,09	1.590,81	17,03	16,91	17,06	16,73
c1355	2.617,81	2.509,32	992,05	973,57	991,16	970,57
c1908	14.804,72	19.451,54	1.703,99	1.738,98	1.717,62	1.727,05
c2670	179.373,22	168.647,98	156.909,14	154.530,60	156.546,58	158.012,38
c3540	214.510,73	198.062,69	103.419,97	101.911,30	103.546,40	103.544,96
c5315	1.982.201,73	1.609.848,62	261.089,51	258.668,01	258.590,58	258.018,69
c6288	48.892,64	22.355,60	821,55	807,76	726,31	826,12
c7552	98.937,07	217.946,48	154.284,93	144.561,17	144.401,50	144.676,80

B.8 Método Híbrido: Aleatório Estrutural e Método ATPG Estrutural Algoritmo-D

Tabela 32 – Validação do Método Híbrido (Aleatório Estrutural e Algoritmo-D) com Limite = 16 e *Backtrack* = 1. Cálculo da Métrica FoM: $(1 - FC) \cdot \#P^2 \cdot TE^{\frac{1}{2}}$

	Backtrack-Limite = 1-16				
Circuitos	#Padrões	#Abortadas	Cobertura	Time (s)	FoM
c17	8	0	100,00%	1,10	0,01
c432	80	5	99,05%	13,40	225,91
c499	56	35	95,38%	21,05	665,76
c880	136	0	100,00%	80,54	16,60
c1355	61	170	89,20%	2.107,08	18.464,84
c1908	205	243	87,07%	9.734,71	536.641,26
c2670	258	176	93,59%	21.921,69	632.424,21
c3540	440	183	94,66%	12.997,23	1.180.469,25
c5315	255	87	98,37%	20.059,24	150.686,39
c6288	46	40	99,48%	3.288,91	638,91
c7552	513	338	95,52%	285.674,99	6.311.136,16

B.9 Método Híbrido: Aleatório Estrutural e Método ATPG baseado em SAT com Compactação

Tabela 33 – Validação do Método Híbrido (Aleatório Estrutural e SAT com Compactação) com Limite = 4 e *Time Out* = 0,01s e 0,1s

Limite = 4	0,01s					0,1s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	6	0	0	100,00%	0,64	6	0	0	100,00%	0,35
c432	74	3	4	98,66%	5,69	75	0	4	99,24%	4,68
c499	70	0	8	98,94%	5,48	63	0	8	98,94%	5,32
c880	112	4	0	99,58%	27,89	115	0	0	100,00%	21,50
c1355	99	9	8	98,92%	61,62	109	0	8	99,49%	84,59
c1908	165	17	7	98,72%	316,43	161	0	7	99,63%	297,67
c2670	157	24	108	95,19%	371,87	166	0	115	95,81%	388,84
c3540	229	101	95	94,28%	2.222,96	261	5	126	96,18%	2.519,34
c5315	202	39	51	98,32%	1.633,94	205	0	59	98,90%	1.511,30
c6288	42	52	2	99,30%	1.635,82	50	38	3	99,47%	1.634,10
c7552	246	141	106	96,73%	3.243,77	321	1	130	98,26%	4.080,51

Tabela 34 – Validação do Método Híbrido (Aleatório Estrutural e SAT com Compactação) com Limite = 4 e *Time Out* = 1s

Limite = 4	1s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	8	0	0	100,00%	0,41
c432	72	0	4	99,24%	6,30
c499	66	0	8	98,94%	6,50
c880	104	0	0	100,00%	30,78
c1355	107	0	8	99,49%	70,63
c1908	179	0	7	99,63%	314,35
c2670	174	0	115	95,81%	398,09
c3540	294	0	131	96,18%	2.549,94
c5315	224	0	59	98,90%	1.707,49
c6288	37	27	7	99,56%	1.429,36
c7552	294	0	131	98,26%	3.592,58

Tabela 35 – Validação do Método Híbrido (Aleatório Estrutural e SAT com Compactação) com Limite = 8 e *Time Out* = 0,01s e 0,1s

Limite = 8	0,01s					0,1s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	6	0	0	100,00%	0,57	7	0	0	100,00%	0,52
c432	73	0	4	99,24%	5,34	80	0	4	99,24%	5,17
c499	73	0	8	98,94%	5,71	62	0	8	98,94%	4,18
c880	107	8	0	99,15%	25,70	118	0	0	100,00%	22,99
c1355	113	0	8	99,49%	72,76	110	0	8	99,49%	61,08
c1908	169	6	7	99,31%	347,73	167	0	7	99,63%	313,63
c2670	168	2	113	95,81%	385,34	169	0	115	95,81%	382,22
c3540	294	55	99	95,51%	3.060,21	285	11	121	96,15%	2.789,10
c5315	230	12	55	98,75%	1.779,03	238	1	58	98,90%	2.122,99
c6288	43	44	2	99,41%	1.809,43	47	33	3	99,54%	1.950,98
c7552	253	210	109	95,77%	4.530,35	342	1	130	98,26%	6.223,69

Tabela 36 – Validação do Método Híbrido (Aleatório Estrutural e SAT com Compactação) com Limite = 8 e *Time Out* = 1s

Limite = 8	1s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	9	0	0	100,00%	0,55
c432	80	0	4	99,24%	5,27
c499	68	0	8	98,94%	7,15
c880	110	0	0	100,00%	33,02
c1355	104	0	8	99,49%	70,89
c1908	180	0	7	99,63%	362,41
c2670	163	0	115	95,81%	441,20
c3540	281	0	131	96,18%	2.736,60
c5315	229	0	59	98,90%	2.267,49
c6288	49	27	7	99,56%	1.875,50
c7552	348	0	131	98,26%	5.863,65

Tabela 37 – Validação do Método Híbrido (Aleatório Estrutural e SAT com Compactação) com Limite = 16 e *Time Out* = 0,01s e 0,1s

Limite = 16	0,01s					0,1s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	6	0	0	100,00%	0,38	9	0	0	100,00%	0,58
c432	74	0	4	99,24%	6,36	83	0	4	99,24%	7,23
c499	70	0	8	98,94%	7,20	69	0	8	98,94%	5,34
c880	117	1	0	99,89%	34,12	117	0	0	100,00%	27,43
c1355	105	0	8	99,49%	62,39	111	0	8	99,49%	67,51
c1908	201	1	7	99,57%	551,33	183	0	7	99,63%	440,20
c2670	179	2	114	95,78%	447,56	179	0	115	95,81%	424,65
c3540	310	50	98	95,68%	3.271,93	292	13	119	96,15%	2.880,53
c5315	228	11	52	98,82%	3.034,86	238	0	59	98,90%	1.981,85
c6288	45	36	2	99,51%	2.493,80	47	31	3	99,56%	2.516,36
c7552	303	196	108	95,97%	6.943,20	376	1	130	98,26%	8.620,72

Tabela 38 – Validação do Método Híbrido (Aleatório Estrutural e SAT com Compactação) com Limite = 16 e *Time Out* = 1s

Limite = 16	1s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	6	0	0	100,00%	0,45
c432	87	0	4	99,24%	8,16
c499	67	0	8	98,94%	5,23
c880	111	0	0	100,00%	30,57
c1355	106	0	8	99,49%	67,38
c1908	204	0	7	99,63%	517,97
c2670	176	0	115	95,81%	449,93
c3540	313	0	131	96,18%	3.473,38
c5315	243	0	59	98,90%	3.348,82
c6288	43	27	7	99,56%	2.578,54
c7552	370	0	131	98,26%	7.984,40

Tabela 39 – Cálculo da Figura de Mérito com a Configuração: $(1 - FC) \cdot \#P^2 \cdot TE^{\frac{1}{2}}$

	Limite								
	4			8			16		
Circuitos	0,01s	0,1s	1s	0,01s	0,1s	1s	0,01s	0,1s	1s
c432	175,78	94,09	100,61	95,26	112,57	113,59	106,79	143,29	167,25
c499	122,24	97,55	118,29	135,68	83,69	131,72	140,04	117,18	109,32
c880	287,91	6,13	6,00	498,75	6,68	6,95	92,91	7,17	6,81
c1355	838,72	566,36	498,71	564,54	490,16	471,99	451,35	524,69	478,03
c1908	6.234,29	1.710,60	2.172,93	3.738,27	1.889,18	2.359,26	4.134,16	2.687,57	3.622,80
c2670	22.888,03	22.802,33	25.349,26	23.249,58	23.431,97	23.419,18	28.692,02	27.707,46	27.572,45
c3540	141.615,10	131.006,66	167.235,59	215.285,21	165.606,08	158.265,68	237.876,84	176.667,52	221.224,81
c5315	27.910,79	18.180,28	23.072,37	28.164,97	29.043,20	27.788,19	34.010,18	28.061,23	38.025,57
c6288	504,63	545,12	232,39	475,06	463,37	466,87	506,33	497,54	421,57
c7552	113.101,25	114.864,97	90.410,42	182.465,61	161.026,20	161.831,43	308.794,81	229.069,82	213.474,02

B.10 Método Híbrido: Aleatório Estrutural, Método ATPG baseado em SAT com Compactação e Algoritmo-D

Tabela 40 – Validação do Método Híbrido (Aleatório Estrutural, SAT com Compactação e Algoritmo-D) com *Backtrack* = 1, Limite = 4 e *Time Out* = 0,001s e 0,01s

	Backtrack = 1 e Limite = 4									
	0,001s					0,01s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	9	0	0	100,00%	0,49	9	0	0	100,00%	0,40
c432	77	0	4	99,24%	6,58	77	0	4	99,24%	5,73
c499	72	3	7	98,68%	5,23	67	3	8	98,55%	9,38
c880	100	0	0	100,00%	29,33	116	0	0	100,00%	26,76
c1355	113	13	6	98,79%	67,30	108	3	8	99,30%	94,11
c1908	179	10	7	99,10%	286,98	179	11	7	99,04%	318,77
c2670	174	8	108	95,78%	385,33	176	13	105	95,70%	378,45
c3540	275	48	94	95,86%	2.498,47	253	54	95	95,65%	2.279,56
c5315	231	16	49	98,79%	1.568,34	269	21	47	98,73%	1.647,16
c6288	44	38	2	99,48%	1.671,02	38	59	2	99,21%	1.484,88
c7552	377	48	107	97,95%	4.410,17	376	52	107	97,89%	3.395,97

Tabela 41 – Validação do Método Híbrido (Aleatório Estrutural, SAT com Compactação e Algoritmo-D) com *Backtrack* = 1, Limite = 4 e *Time Out* = 0,1s

	Backtrack = 1 e Limite = 4				
	0,1s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	7	0	0	100,00%	0,29
c432	77	0	4	99,24%	5,17
c499	67	0	8	98,94%	4,95
c880	107	0	0	100,00%	37,03
c1355	113	0	8	99,49%	72,13
c1908	171	0	7	99,63%	338,62
c2670	177	0	115	95,81%	421,59
c3540	261	6	125	96,18%	3.035,43
c5315	236	0	59	98,90%	1.835,32
c6288	53	34	3	99,52%	1.705,52
c7552	326	1	130	98,26%	4.241,17

Tabela 42 – Validação do Método Híbrido (Aleatório Estrutural, SAT com Compactação e Algoritmo-D) com *Backtrack* = 1, Limite = 8 e *Time Out* = 0,001s e 0,01s

	Backtrack = 1 e Limite = 8									
	0,001s					0,01s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	7	0	0	100,00%	0,33	8	0	0	100,00%	0,36
c432	69	1	4	99,05%	4,51	73	0	4	99,24%	5,18
c499	62	0	8	98,94%	5,15	71	0	8	98,94%	5,14
c880	112	0	0	100,00%	23,37	107	0	0	100,00%	25,34
c1355	110	3	8	99,30%	67,34	103	0	8	99,49%	61,62
c1908	170	12	7	98,99%	263,30	166	4	7	99,41%	301,54
c2670	171	16	103	95,67%	386,25	160	3	112	95,81%	402,77
c3540	284	45	93	95,97%	2.551,72	278	39	100	95,95%	2.629,60
c5315	230	23	44	98,75%	1.517,74	243	6	53	98,90%	2.403,54
c6288	44	39	2	99,47%	1.848,04	47	40	2	99,46%	1.975,45
c7552	423	61	102	97,84%	5.249,56	399	32	114	98,07%	5.507,99

Tabela 43 – Validação do Método Híbrido (Aleatório Estrutural, SAT com Compactação e Algoritmo-D) com *Backtrack* = 1, Limite = 8 e *Time Out* = 0,1s

	Backtrack = 1 e Limite = 8				
	0,1s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	6	0	0	100,00%	0,39
c432	65	0	4	99,24%	4,53
c499	63	0	8	98,94%	4,46
c880	119	0	0	100,00%	29,58
c1355	105	0	8	99,49%	51,95
c1908	170	0	7	99,63%	273,71
c2670	172	0	115	95,81%	387,02
c3540	288	6	125	96,18%	2.587,70
c5315	219	0	59	98,90%	1.941,38
c6288	55	33	3	99,54%	2.189,05
c7552	366	1	130	98,26%	6.104,62

Tabela 44 – Validação do Método Híbrido (Aleatório Estrutural, SAT com Compactação e Algoritmo-D) com *Backtrack* = 1, Limite = 16 e *Time Out* = 0,001s e 0,01s

	Backtrack = 1 e Limite = 16									
	0,001s					0,01s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	8	0	0	100,00%	0,39	7	0	0	100,00%	0,40
c432	69	1	4	99,05%	5,34	75	0	4	99,24%	6,31
c499	59	26	0	96,57%	5,08	70	0	8	98,94%	6,06
c880	111	0	0	100,00%	25,90	114	0	0	100,00%	26,36
c1355	109	80	0	94,92%	54,29	105	1	8	99,43%	54,69
c1908	229	109	5	93,93%	320,32	174	2	7	99,52%	334,49
c2670	363	77	68	94,72%	337,80	189	10	107	95,74%	439,73
c3540	414	126	60	94,57%	2.406,31	302	39	97	96,03%	2.784,33
c5315	281	53	33	98,39%	2.752,35	240	8	54	98,84%	2.677,21
c6288	52	32	2	99,56%	2.621,26	46	36	2	99,51%	2.178,03
c7552	650	313	22	95,56%	8.075,43	398	50	106	97,93%	8.184,61

Tabela 45 – Validação do Método Híbrido (Aleatório Estrutural, SAT com Compactação e Algoritmo-D) com *Backtrack* = 1, Limite = 16 e *Time Out* = 0,1s

	Backtrack = 1 e Limite = 16				
	0,1s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	7	0	0	100,00%	0,37
c432	80	0	4	99,24%	6,78
c499	71	0	8	98,94%	5,73
c880	111	0	0	100,00%	27,47
c1355	109	0	8	99,49%	64,57
c1908	207	0	7	99,63%	555,66
c2670	180	0	115	95,81%	481,12
c3540	312	5	126	96,18%	2.837,71
c5315	230	0	59	98,90%	2.398,15
c6288	52	35	3	99,51%	2.814,05
c7552	363	1	130	98,26%	7.651,96

Tabela 46 – Validação do Método Híbrido (Aleatório Estrutural, SAT com Compactação e Algoritmo-D) com *Backtrack* = 5, Limite = 4 e *Time Out* = 0,001s e 0,01s

	Backtrack = 5 e Limite = 4									
	0,001s					0,01s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	8	0	0	100,00%	0,37	8	0	0	100,00%	0,38
c432	83	2	2	99,24%	5,02	62	0	4	99,24%	4,32
c499	63	35	0	95,38%	5,54	65	1	7	98,94%	4,84
c880	107	0	0	100,00%	23,82	109	0	0	100,00%	21,98
c1355	100	4	7	99,30%	61,17	106	11	5	98,98%	64,99
c1908	189	4	7	99,41%	339,37	175	11	7	99,04%	442,22
c2670	173	14	104	95,70%	373,85	173	14	103	95,74%	440,86
c3540	284	44	92	96,03%	2.431,23	286	43	91	96,09%	2.497,65
c5315	647	87	33	97,76%	823,94	250	13	48	98,86%	1.604,72
c6288	45	60	2	99,20%	1.635,20	54	77	2	98,98%	1.755,25
c7552	357	39	109	98,04%	3.983,37	410	38	108	98,07%	5.250,89

Tabela 47 – Validação do Método Híbrido (Aleatório Estrutural, SAT com Compactação e Algoritmo-D) com *Backtrack* = 5, Limite = 4 e *Time Out* = 0,1s

	Backtrack = 5 e Limite = 4				
	0,1s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	7	0	0	100,00%	0,38
c432	69	0	4	99,24%	5,44
c499	67	0	8	98,94%	5,77
c880	101	0	0	100,00%	23,26
c1355	109	0	8	99,49%	67,14
c1908	166	0	7	99,63%	320,85
c2670	171	0	115	95,81%	422,47
c3540	244	5	126	96,18%	2.180,03
c5315	216	0	59	98,90%	1.827,67
c6288	50	32	3	99,55%	1.981,37
c7552	345	1	130	98,26%	4.769,80

Tabela 48 – Validação do Método Híbrido (Aleatório Estrutural, SAT com Compactação e Algoritmo-D) com *Backtrack* = 5, Limite = 8 e *Time Out* = 0,001s e 0,01s

	Backtrack = 5 e Limite = 8									
	0,001s					0,01s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	9	0	0	100,00%	0,51	7	0	0	100,00%	0,35
c432	73	0	4	99,24%	5,12	78	0	4	99,24%	6,70
c499	71	2	7	98,81%	6,10	64	2	7	98,81%	5,38
c880	119	0	0	100,00%	23,98	106	0	0	100,00%	22,89
c1355	111	4	7	99,30%	68,26	112	13	3	98,98%	69,53
c1908	171	4	7	99,41%	287,13	168	7	7	99,25%	309,75
c2670	172	12	107	95,67%	392,28	165	12	107	95,67%	333,82
c3540	256	50	92	95,86%	2.298,68	279	46	93	95,95%	2.448,83
c5315	256	15	46	98,86%	1.800,98	255	12	47	98,90%	1.867,89
c6288	36	37	2	99,50%	1.653,82	48	40	2	99,46%	2.051,44
c7552	412	53	101	97,96%	4.705,76	421	46	101	98,05%	4.766,32

Tabela 49 – Validação do Método Híbrido (Aleatório Estrutural, SAT com Compactação e Algoritmo-D) com *Backtrack* = 5, Limite = 8 e *Time Out* = 0,1s

	Backtrack = 5 e Limite = 8				
	0,1s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	7	0	0	100,00%	0,40
c432	76	0	4	99,24%	5,38
c499	71	0	8	98,94%	5,06
c880	110	0	0	100,00%	20,18
c1355	106	0	8	99,49%	59,13
c1908	170	0	7	99,63%	311,29
c2670	168	0	115	95,81%	362,09
c3540	294	4	127	96,18%	2.810,47
c5315	217	0	59	98,90%	1.832,25
c6288	50	32	3	99,55%	1.821,84
c7552	339	1	130	98,26%	5.444,52

Tabela 50 – Validação do Método Híbrido (Aleatório Estrutural, SAT com Compactação e Algoritmo-D) com *Backtrack* = 5, Limite = 16 e *Time Out* = 0,001s e 0,01s

Backtrack = 5 e Limite = 16										
	0,001s					0,01s				
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	6	0	0	100,00%	0,37	8	0	0	100,00%	0,62
c432	71	0	4	99,24%	5,42	78	0	4	99,24%	6,35
c499	69	1	8	98,81%	6,35	69	0	8	98,94%	5,93
c880	106	0	0	100,00%	22,55	113	0	0	100,00%	31,18
c1355	106	7	1	99,49%	57,91	114	0	8	99,49%	78,28
c1908	181	4	6	99,47%	371,01	200	0	7	99,63%	492,35
c2670	178	8	109	95,74%	465,95	172	2	113	95,81%	448,12
c3540	309	43	91	96,09%	3.157,46	327	33	100	96,12%	3.505,27
c5315	250	10	49	98,90%	2.750,21	248	4	55	98,90%	3.307,06
c6288	47	32	2	99,56%	2.111,55	48	38	2	99,48%	2.354,86
c7552	458	62	101	97,84%	6.090,63	371	31	114	98,08%	5.418,45

Tabela 51 – Validação do Método Híbrido (Aleatório Estrutural, SAT com Compactação e Algoritmo-D) com *Backtrack* = 5, Limite = 16 e *Time Out* = 0,1s

Backtrack = 5 e Limite = 16					
0,1s					
Circuitos	#Padrões	#Abortadas	#unSAT	Cobertura	Tempo (s)
c17	7	0	0	100,00%	0,67
c432	82	0	4	99,24%	6,50
c499	67	0	8	98,94%	5,57
c880	114	0	0	100,00%	26,85
c1355	109	0	8	99,49%	69,36
c1908	196	0	7	99,63%	472,66
c2670	172	0	115	95,81%	485,04
c3540	293	5	126	96,18%	2.520,14
c5315	219	0	59	98,90%	1.983,80
c6288	53	37	3	99,48%	2.378,92
c7552	376	1	130	98,26%	7.495,92

Tabela 52 – Cálculo da Figura de Mérito com a Configuração: $(1 - FC) \cdot \#P^2 \cdot TE^{\frac{1}{2}}$

	Backtrack = 1								
	Limite = 4			Limite = 8			Limite = 16		
Circuitos	0,001s	0,01s	0,1s	0,001s	0,01s	0,1s	0,001s	0,01s	0,1s
c432	117,60	109,73	104,27	97,46	93,77	69,58	106,06	109,24	128,92
c499	157,54	200,92	106,46	92,96	121,79	89,33	269,90	128,55	128,61
c880	5,42	6,96	6,97	6,06	5,76	7,70	6,27	6,67	6,46
c1355	1.274,93	802,13	562,08	703,88	431,62	411,85	4.458,22	474,36	494,83
c1908	4.964,88	5.537,58	2.058,16	4.788,92	2.849,04	1.828,82	57.037,78	2.707,69	3.863,46
c2670	25.156,04	25.945,73	26.994,12	24.952,74	21.559,85	24.423,22	128.078,82	31.978,94	29.822,78
c3540	156.964,57	133.141,69	143.800,34	164.426,58	161.091,88	161.663,20	457.033,70	191.409,25	198.683,61
c5315	25.886,87	37.620,23	26.551,95	26.014,62	32.214,80	23.515,87	67.005,20	34.837,01	28.827,74
c6288	416,67	443,87	565,88	448,92	542,35	672,13	621,60	494,45	718,21
c7552	194.719,58	174.330,41	120.780,94	281.178,32	229.664,55	182.646,66	1.688.442,77	297.532,51	201.150,00

Tabela 53 – Cálculo da Figura de Mérito com a Configuração: $(1 - FC) \cdot \#P^2 \cdot TE^{\frac{1}{2}}$

	Backtrack = 5								
	Limite = 4			Limite = 8			Limite = 16		
Circuitos	0,001s	0,01s	0,1s	0,001s	0,01s	0,1s	0,001s	0,01s	0,1s
c432	119,39	61,81	85,84	93,30	121,77	103,64	90,76	118,59	132,61
c499	432,44	99,04	114,88	149,09	113,80	120,86	143,65	123,50	112,90
c880	5,59	5,57	4,92	6,93	5,38	5,44	5,34	7,13	6,73
c1355	554,43	929,83	504,57	721,63	1.073,67	447,80	443,18	595,97	512,84
c1908	3.918,06	6.234,08	1.887,99	2.950,11	3.750,85	1.950,36	3.421,45	3.394,92	3.194,60
c2670	24.915,65	26.828,16	25.221,21	25.441,76	21.597,97	22.537,54	29.198,22	26.280,40	27.341,41
c3540	158.174,94	160.203,81	106.507,36	130.472,31	156.576,66	175.571,01	210.262,28	246.254,30	165.126,37
c5315	270.718,60	28.797,40	22.195,94	31.989,51	31.273,28	22.430,01	36.473,70	39.358,74	23.771,40
c6288	663,77	1.258,45	514,12	270,69	576,46	492,99	455,77	588,66	721,34
c7552	158.487,18	236.774,86	143.452,65	238.671,03	239.467,40	147.978,83	355.060,00	195.593,86	213.603,61