UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

LEONARDO DROVES SILVEIRA

Uma nova estrutura de dados para ferramentas de síntese lógica

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof. Dr. André Inácio Reis

Porto Alegre
2022

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos André Bulhões Mendes
Vice-Reitora: Profª. Patricia Helena Lucas Pranke
Pró-Reitoria de Graduação: Profª. Cíntia Inês Boll
Diretora do Instituto de Informática: Profª. Carla Maria Dal Sasso Freitas
Diretora da Escola de Engenharia: Profª. Carla Schwengber Ten Caten
Coordenador do Curso de Engenharia de Computação: Prof. Walter Fetter Lages
Bibliotecário-chefe do Instituto de Informática: Alexsander Borges Ribeiro
Bibliotecária-chefe da Escola de Engenharia: Rosane Beatriz Allegretti Borges

"Be ashamed to die until you have won some victory for humanity."

— HORACE MANN

AGRADECIMENTOS

Gostaria de agradecer e dedicar esse trabalho aos meus pais e irmãos por todo o amor e suporte incondicional me dado, por todos os sacrifícios realizados e por toda confiança depositada. A vocês meu eterno amor e carinho.

Quero agradecer ao meu orientador, André Inácio Reis, por todo o conhecimento transferido, por todas as oportunidades e pelo suporte. O senhor transformou minha graduação em uma experiência única, motivante e divertida. Serei eternamente grato.

Quero agradecer também ao Alcides Silveira Costa, por toda experiência compartilhada, por todos os conselhos e pelo apoio.

Gostaria de agradecer também a todo o corpo docente do instituto de informática e demais departamentos pelo empenho e dedicação em me proporcionar um ensino de excelência.

Por fim mas não menos importante, gostaria de agradecer aos meus amigos, colegas e familiares por todos os momentos de alegria e descontração proporcionados durante estes longos 5 anos.

A todos os meus mais sinceros agradecimentos.

RESUMO

Este trabalho propõe uma nova estrutura de dados para as ferramentas de síntese lógica. Esta nova estrutura pretende unir duas estruturas bem conhecidas na área de síntese lógica, o AIG (And-Inverter-Graph) com o BDD (Binary Decision Diagram), formando uma estrutura híbrida e canônica que apresente um bom desempenho na síntese lógica quando utilizado o método de composição funcional.

Palavras-chave: Síntese lógica. Estrutura de dados. AIG. BDD.

ABSTRACT

This work presents a novel data structure to logic synthesis tools. This new data structure intends to join two well-know data structures in the logic synthesis field, AIG (And-Inverter-Graph) with BDD (Binary Decision Diagram), merging into a hybrid and canonical structure that shows a good performance when running logic synthesis with the composition method.

Keywords: logic synthesis, data structures, AIG, BDD.

LISTA DE ABREVIATURAS E SIGLAS

AIG	And-Inverter Graph
BDD	Binary Decision Diagram
DAG	Direct Aciclic Graph
EDA	Electronic Design Automation
IWLS	International Workshop on Logic and Synthesis)
PDS	Produtos de Somas
POS	Product of Sums
ROBDD	Reduced Ordered Binary Decision Diagram
SDP	Somas de Produtos
SOP	Sum of Products
TCC	Trabalho de Conclusão de Curso

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A área de síntese de hardware é muito dependente de ferramentas computacionais eficientes que auxiliem essa síntese. As ferramentas de auxilio a síntese de circuitos integrados são convencionalmente chamadas de ferramentas de EDA, onde a sigla EDA significa *Electronic Design Automation* em inglês. A área de EDA se encontra no cruzamento entre as áreas de microeletrônica e da ciência da computação. O conhecimento de microeletrônica que é importante, é o conhecimento dos objetos (circuitos integrados, redes de portas lógicas, tabelas verdades e etc) a serem modelados. O conhecimento de ciência da computação que é importante inclui complexidade e escalabilidade através de algorítimos eficientes. Isso é importante por causa da escala de integração atual, onde circuitos integrados são compostos de um grande número de primitivas, o que aumenta a complexidade computacional, devido ao grande tamanho do espaço de soluções.

Em recente palestra do pesquisador Alan Mishchenko (MISHCHENKO, 2022), foram enfatizados problemas de pesquisa ainda não resolvidos na área da síntese lógica. Ao revisar o estado da arte nesta palestra, ficaram claras algumas tendências. Um dos pontos levantados é que a estrutura de dados do tipo AIG (*And-Inverter Graphs*, em inglês ou Grafo de Ands e inversores em português) é uma estrutura central em ferramentas de síntese lógica modernas. Outro ponto discutido por Mishchenko é que progressos em qualidade ou em tempo de execução, ainda que em sub-circuitos pequenos, são bem vindos. Este é o caso do concurso do IWLS 2022 (International Workshop on Logic and Synthesis) que tratava apenas de circuitos com 16 entradas ou menos. Assim sendo, uma contribuição possível é propor versões mais eficientes de estruturas de dados do tipo AIG.

Os AIGs são conhecidos por serem estruturas de dados semi-canônicas, o que significa que podem existir vários AIGs diferentes para uma mesma função Booleana. Isto por si só não é um defeito, pois diferentes AIGs podem corresponder a diferentes implementações de uma mesma função Booleana, que podem ser úteis em diferentes contextos de função de custo e objetivos de implementação. Um outro tipo de grafo são os BDDs (*Binary Decision Diagram* em inglês ou diagramas de decisão binários). BDDs são estruturas de dados canônicas em sua forma reduzida e ordenada (ROBDD). Na prática isso significa que existe uma correspondência única entre nodos de BDDs e funções Booleanas. Este trabalho pretende propor uma estrutura de dados que integre as funcionalidades de AIGs e BDDs em uma mesma estrutura de dados. Assim as múltiplas soluções existentes em um AIG para uma mesma função Booleana poderão ser associadas

a função Booleana representada, permitindo a escolha da estrutura com os custos mais alinhados aos objetivos durante a síntese.

Este trabalho está organizado da seguinte forma: No capítulo 2 apresentamos alguns dos conceitos básicos necessários para um bom entendimento dos capítulos seguintes. Já no capítulo 3 realizamos uma revisão bibliográfica contendo alguns artigos relacionados à proposta deste trabalho. Nos capítulos 4 e 5, apresentaremos as duas estruturas de dados importantes para este trabalho, o AIG e o BDD, respectivamente. No capítulo 6 realizamos uma comparação entre estas estruturas de dados. No capítulo 7 iremos propor uma nova estrutura de dados, o AIG completamente canônico. No capítulo 8 apresentamos os resultados preliminares obtidos. Por fim, no capítulo 9, concluímos este trabalho.

2 CONCEITOS BÁSICOS

Neste capítulo apresentaremos alguns dos principais conceitos necessários para um bom entendimento dos próximos capítulos. Um leitor com mais experiência na área de síntese lógica pode seguir para o próximo capítulo sem a necessidade da leitura deste. Porém, para um leitor com pouca experiência na área, a leitura deste capítulo pode ser muito esclarecedora e facilitará a compreensão dos próximos capítulos.

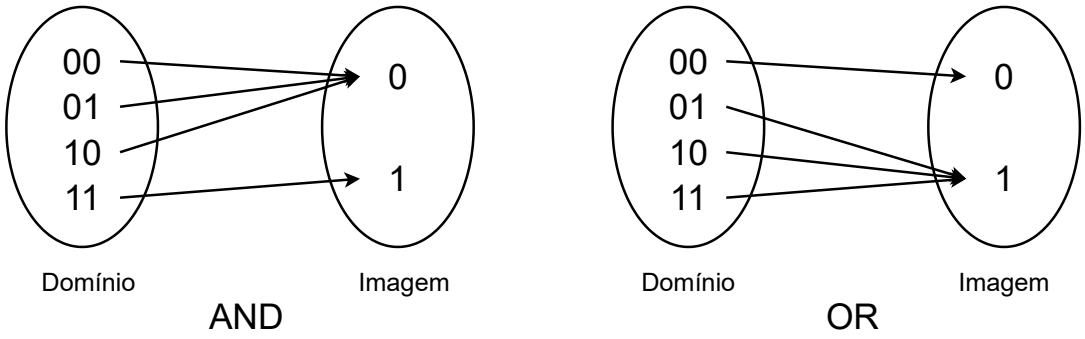
2.1 Funções Booleanas

Em matemática, uma função é uma relação entre elementos de 2 conjuntos. A função $f : A \rightarrow B$ relaciona elementos do conjunto A (domínio) com elementos do conjunto B (imagem ou contradomínio).

Uma função Booleana $g : C \rightarrow D$ relaciona elementos de um conjunto arbitrário C com os elementos do conjunto D, chamado de domínio booleano, que consiste de apenas 2 elementos, chamados booleanos, representando verdadeiro e falso. O domínio booleano é muitas vezes escrito como $D = \{0, 1\}$.

Na figura 2.1 podemos ver as funções Booleanas And e Or de 2 entradas (variáveis):

Figura 2.1: Funções Booleanas And e Or



Fonte: Autor e Orientador

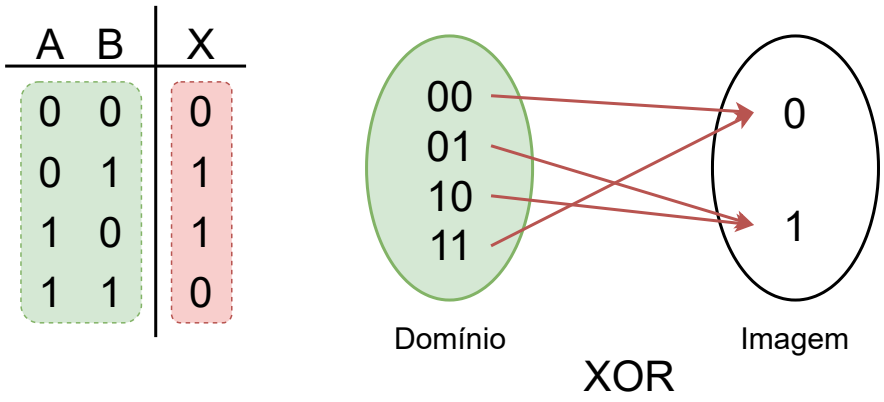
2.1.1 Tabela Verdade

Uma das formas de representar uma função Booleana é utilizando a tabela verdade. A tabela verdade define uma função Booleana através das entradas (variáveis) e

saída da função.

Um exemplo pode ser visto na figura 2.2. Em verde está representado o conjunto domínio, definido pelas variáveis de entrada A e B. Na coluna em vermelho está a função Booleana de saída. Esta tabela verdade define a função Booleana Xor, também conhecida como "Ou exclusivo".

Figura 2.2: Tabela verdade para a função Booleana Xor



Fonte: Autor e Orientador

2.2 Equações Booleanas

Uma outra forma de representar uma função Booleana é utilizando equações Booleanas. Essas equações descrevem uma função Booleana utilizando-se dos literais e operações Booleanas – and (·), or (+), not (overline) e etc.

Por exemplo: a função Booleana Xor representada anteriormente na tabela verdade pode ser escrita como a equação Booleana: $xor(a,b) = \overline{a} \cdot b + a \cdot \overline{b}$.

Note que uma função Booleana pode ser representada por mais de uma equação. A função Xor pode ser rescrita também desta forma: $xor(a,b) = (\overline{a} + \overline{b}) \cdot (a + b)$ e outras mais.

2.2.1 Variáveis

As variáveis de uma equação Booleana indicam o tamanho do conjunto domínio da função Booleana que ela representa. No exemplo da figura 2.2 podemos identificar 2 variáveis (A e B). O tamanho do conjunto domínio será 2^n, sendo n o número de variáveis

da função sendo representada.

2.2.2 Literais

Muitas vezes confundido com o conceito de variável, os literais são a representação das variáveis na equação Booleana. Por exemplo, na equação Booleana 2.1 temos 7 literais e a função Booleana depende de 3 variáveis (a, b, c).

$$f_1(a, b, c) = a \cdot \overline{b} + (\overline{a} \cdot c + b \cdot (a + c)) \tag{2.1}$$

Definimos literal positivo x como a representação da variável x e literal negativo $\overline{x}$ como a representação negada da variável x.

2.2.3 Níveis Lógicos

Toda a equação Booleana possuí um determinado nível lógico. O nível lógico é definido pela quantidade máxima de portas lógicas entre uma entrada e uma saída. Note que esta definição é ambígua. Esta tem sido uma discussão recorrente no grupo LogiCS e é uma fonte de ambiguidade em discussões sobre síntese lógica. Consideremos, por exemplo, somas de produto (SDP) e produtos de somas (PDS) que são consideradas formas a dois níveis em termos de equação, porém em termos de circuito talvez os dois níveis não se apliquem. Isto é talvez uma ambiguidade e uma contradição, mas não se deve fugir desta situação e sim enfrentá-la para produzir clareza.

Assim sendo, vamos definir o número de níveis lógicos de uma equação ou de um circuito como o número de operadores (ou portas lógicas) que necessitam ser avaliados em sequência durante a propagação de uma entrada até a saída. Assim sendo, uma equação ou circuito pode ser expressa como uma árvore (de operadores) e o número de níveis lógicos é a profundidade máxima da árvore. A partir desta definição, as seguintes questões aparecem.

Questão 1: Qual o tamanho do nodo da árvore? Esta questão reflete a possibilidade de esconder níveis lógicos dentro de um nodo da árvore, permitindo nodos com operações complexas ou com um grande número de entradas. É preciso entender que níveis lógicos podem ser escondidos dentro de um nodo da árvore e que isto deve ser evitado caso não resulte em ganhos reais de diminuição de atraso.

Questão 2: Inversores contam como níveis lógicos? Esta é uma pergunta sujeita a debate. A resposta geralmente aceita é que inversores não contam como níveis lógicos. Porém o bom senso manda que o número de níveis lógicos deve ser uma medida aproximada do atraso do circuito final. Assim, se o número de níveis lógicos contando inversores tiver uma correspondência direta com o atraso final, esta deve ser a medida de níveis lógicos considerada.

2.2.4 Formas a 2 níveis: SOP e POS

Duas formas de equações Booleanas interessantes e amplamente utilizadas são a soma de produtos (SOP) e o produto das somas (POS). Essas equações possuem 2 níveis lógicos.

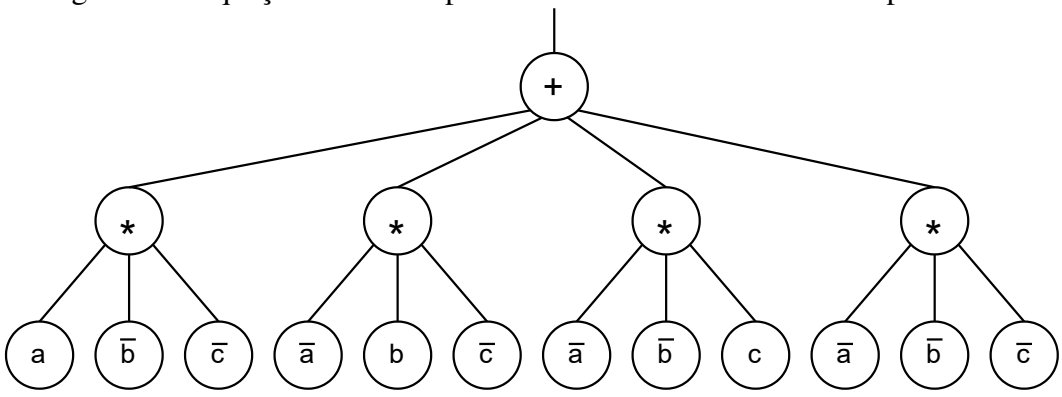
As equações 2.2 e 2.3 escritas nas formas SOP e POS respectivamente, representam a mesma função Booleana.

$$f_2(a,b,c) = (a \cdot \overline{b} \cdot \overline{c}) + (\overline{a} \cdot b \cdot \overline{c}) + (\overline{a} \cdot \overline{b} \cdot c) + (\overline{a} \cdot \overline{b} \cdot \overline{c}) \tag{2.2}$$

$$f_2(a,b,c) = (a + \overline{b} + \overline{c}) \cdot (\overline{a} + b + \overline{c}) \cdot (\overline{a} + \overline{b} + c) \cdot (\overline{a} + \overline{b} + \overline{c}) \tag{2.3}$$

Podemos verificar na figura 2.3 que a função escrita na forma SOP é uma função de 2 níveis já que a a equação decomposta na árvore de operadores possui profundidade máxima igual a 2.

Figura 2.3: Equação 2.2 decomposta na forma de uma árvore de operadores.



Fonte: Autor e Orientador

2.2.5 Formas multinível: forma fatorada

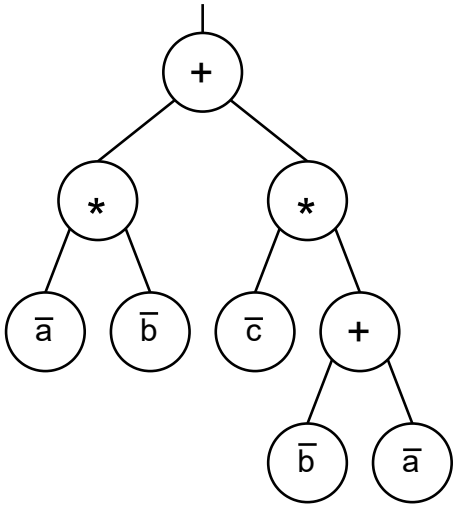
Uma equação Booleana pode ser fatorada. A fatoração de uma equação tem como objetivo a redução do número de literais. O número de literais em uma equação é proporcional ao custo da implementação desta função em um circuito integrado. Quando realizamos a fatoração de uma equação podemos alterar o nível lógico da mesma, a esta nova função, fatorada, damos o nome de forma multinível.

Como exemplo, fatoramos a equação 2.2 e obtivemos a equação 2.4.

$$f_2(a, b, c) = \overline{a} \cdot \overline{b} + (\overline{c} \cdot (\overline{b} + \overline{a})) \tag{2.4}$$

Reduzimos na equação fatorada 2.4 o número de literais de 12 para 5, porém, o nível lógico da equação aumentou de 2 para 3, se tornando uma equação multinível. Essas características podem ser vistas na figura 2.4.

Figura 2.4: Equação 2.4 decomposta na forma de uma árvore de operadores.



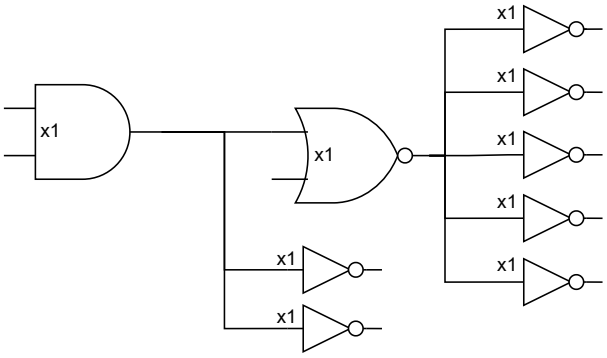
Fonte: Autor e Orientador

2.2.6 Fanout

De modo geral, evitando por enquanto uma definição mais precisa, podemos dizer que o fanout de um circuito lógico representa o número de elementos que usam o sinal gerado por este circuito lógico. Assim, se um circuito está representado como um conjunto de portas lógicas o fanout de cada uma das portas lógicas do circuito pode ser considerado como o número de entradas de outras portas lógicas conectado na saída da porta que se

quer saber o fanout. Isto é melhor explicado através do desenho na Figura 2.5.

Figura 2.5: Circuito com exemplos de fanout. A porta and tem fanout 3 e a porta nor tem fanout 5.



Fonte: Autor e Orientador

Em níveis de descrição menos detalhados, o conceito de fanout pode ser considerado de modo diferente. Em um grafo do tipo AIG (*And-Inverter-Graph*) ou BDD, pode ser o número de arestas que sai (ou chega, dependendo da direção das arestas) em um determinado nodo. Em um conjunto de equações, o fanout de uma variável pode ser considerado como o número total de literais que existem para aquela variável.

Em níveis de descrição do circuito mais detalhadas, o conceito de fanout pode considerar que o peso da carga de entrada pode ser diferente. Bibliotecas de células podem ter células com diferentes tamanhos, que resultem em diferentes capacitâncias de entrada. Além disso o roteamento (capacitâncias de fios) podem contribuir significativamente para o fanout.

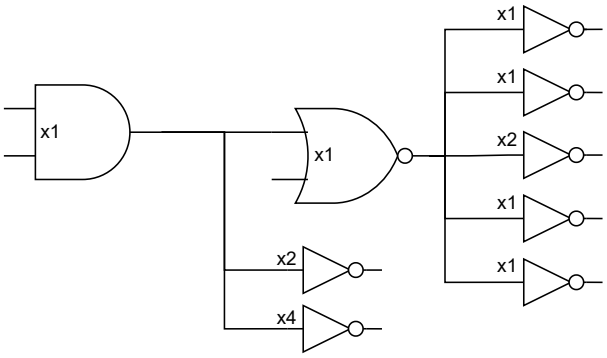
2.2.7 Fanin

O conceito de fanin pode ter duas interpretações. A seguir discutimos estas interpretações.

Em uma primeira interpretação mais comum do fanin, usada nos níveis mais detalhados da descrição de circuitos digitais, o fanin pode ser considerado como uma medida da capacitância de entrada de cada célula. Assim, o circuito na figura 2.6 possui portas com fanout diferentes do circuito da Figura 2.5, apesar de os dois circuitos terem a mesma topologia. Isto acontece porque o fanin das portas mudou, conforme indicado na Figura 2.6.

Em uma segunda interpretação menos comum da palavra fanin, usada nos níveis menos detalhados da descrição de circuitos digitais, o fanin pode ser considerado como

Figura 2.6: Circuito com exemplos de fanin. A porta and agora tem fanout 7 e a porta nor tem fanout 6, pois o fanin (das entradas) das portas na saída aumentou.



Fonte: Autor e Orientador

o número de filhos de um nodo em um grafo. Considere a figura 2.3, podemos dizer que o nodo raiz tem fanin 4, pois o nodo raiz tem 4 filhos. Estas definição de fanin é muito diferente da anterior e estas duas definições distintas não devem ser confundidas apesar do uso da mesma palavra para conceitos diferentes.

3 REVISÃO BIBLIOGRÁFICA

Neste capítulo é feita uma breve revisão bibliográfica dos trabalhos relacionados. Começamos com uma revisão dos trabalhos do grupo LogiCS e depois focamos especialmente nos trabalhos de composição funcional.

3.1 Trabalhos do LogiCS

Esta seção descreve o contexto mais amplo deste trabalho de conclusão (TCC). Vamos descrever este contexto geral a partir dos trabalhos do grupo LogiCS. Trata-se de um texto relativamente padrão usado dentro do grupo LogiCS com permissão do orientador, mas vamos adequá-lo ao escopo deste TCC. O TCC foca no tópico de síntese lógica (REIS; DRECHSLER, 2018) de circuitos digitais (WAGNER; REIS; RIBAS, 2006). A síntese lógica transforma uma descrição lógica inicial do circuito em uma rede de primitivas lógicas tais como células de bibliotecas (TOGNI et al., 2002). A etapa de síntese lógica que cria uma rede de células em uma determinada tecnologia é conhecida como mapeamento tecnológico (REIS, 1999). A síntese lógica é uma etapa importante no fluxo de projeto de circuitos digitais tais como microprocessadores (ROSA et al., 2003). As células da biblioteca são feitas de transistores, e a síntese lógica pode ser usada para sintetizar redes de transistores para as células da biblioteca (JUNIOR et al., 2006; SILVA; REIS; RIBAS, 2009; ROSA et al., 2007; BUTZEN et al., 2010a; BUTZEN et al., 2012; ROSA et al., 2009).

A síntese eficiente de redes de transistores pode ser importante para otimizar custos relacionados a área (POLI et al., 2003), variabilidade (SILVA; REIS; RIBAS, 2009; BUTZEN et al., 2010a; BUTZEN et al., 2012) e potência (BUTZEN et al., 2010b). Métodos propostos no grupo LogiCS, tais como síntese baseada em cortes KL (MACHADO et al., 2012) e síntese baseada em composição funcional (MARTINS; RIBAS; REIS, 2012) são usados em novas tecnologias (NEUTZLING et al., 2013; MARRANGHELLO et al., 2015; NEUTZLING et al., 2015), circuitos robustos (GOMES et al., 2014; GOMES et al., 2015) e circuitos assíncronos (MOREIRA et al., 2014).

No escopo deste trabalho será de especial importância a revisão do método de composição funcional, pois a estrutura que estamos propondo visa a aplicação em composição funcional.

3.2 Composição funcional

A ideia da composição funcional foi apresentada inicialmente em (REIS et al., 2009) e (MARTINS et al., 2010). Em métodos de síntese lógica, usualmente, se deseja sintetizar uma estrutura (e.g.uma equação) a partir de uma funcionalidade (e.g. uma tabela verdade). Por exemplo, o método de Quine-McCluskey gera uma estrutura (SOP) a partir de uma funcionalidade (tabela-verdade). No conceito de composição funcional, não se faz a conversão entre funcionalidades e estruturas (ou vice e versa). Ao invés disso se trabalha com duplas que agregam uma função e uma estrutura garantindo sempre a equivalência, sem necessidade de conversão. Isto é feito através da criação de pares elementares, que são combinados depois. Por exemplo, pode-se criar pares que representam ao mesmo tempo a funcionalidade e a estrutura de literais, que são combinados de modo heurístico até se chegar na funcionalidade desejada. Uma vez alcançado o par com a funcionalidade desejada, a estrutura correspondente neste par representa o resultado da síntese.

3.2.1 Funcionalidade

No artigo original sobre a composição funcional (REIS et al., 2009), a funcionalidade era representada como tabelas-verdade na forma de inteiros. A combinação era feita através de operações *bitwise* nestes inteiros, o que é muito rápido de calcular.

3.2.2 Estrutura

No artigo original sobre a composição funcional (REIS et al., 2009), a estrutura era representada como cadeias de caracteres que representavam equações. A combinação era feita através de concatenações destas cadeias de caracteres, o que é relativamente rápido de calcular.

3.2.3 Par Funcionalidade e Estrutura

Deste modo o par {funcionalidade, estrutura} usado no artigo original era um par {inteiro, string}. Pares diferentes podem ser usados, para se obter algoritmos mais eficientes ou voltados a outros tipos de estrutura. No caso deste TCC, a intenção inicial

era propor uma inovação para representar o par do tipo {BDD, AIG} para minimização de AIGs. A inovação pretendida será discutida após a revisão de AIGs e BDDs nos próximos capítulos.

3.3 Contribuição deste capítulo

Este capítulo fez uma revisão bibliográfica inicial focada em artigos do LogiCS, principalmente no método de composição funcional. A intenção inicial deste trabalho era focar em desenvolver uma nova versão do método de composição funcional usando uma nova forma de representar pares do tipo {BDD, AIG}. Porém esta estrutura de dados mista entre AIGs e BDDs se demonstrou suficientemente rica em detalhes para justificar um TCC por si só. Nos próximos dois capítulos iremos revisar AIGs e BDDs, formando a base para a proposta híbrida apresentada aqui.

4 REVISÃO: AIGS COMO ESTRUTURA SEMI-CANÔNICA

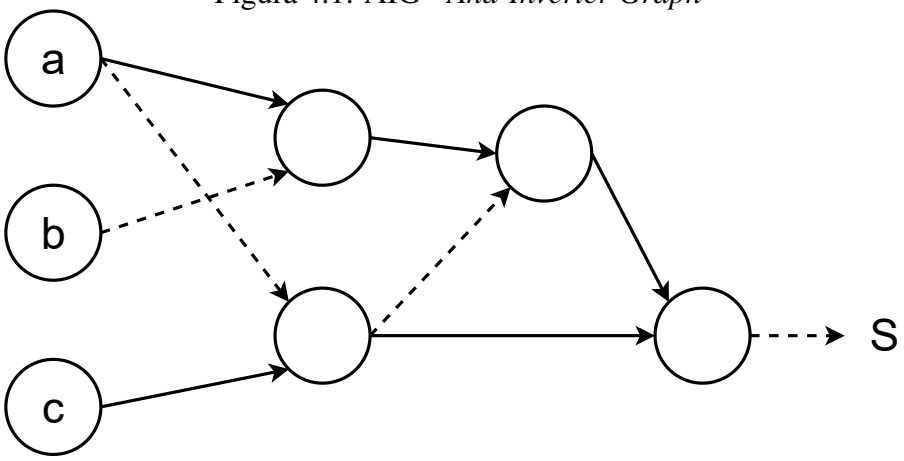
Neste capítulo iremos revisar os AIGs (*And-Inverter-Graphs*, em inglês) como estrutura de dados semi-canônicas para a representação de funções Booleanas. AIGs são muito usados em ferramentas de síntese modernas, tais como o ABC de Berkeley.

4.1 O que é um AIG

O AIG (*And-Inverter Graph* em Inglês) é um grafo acíclico e direto (DAG, em inglês, *Direct Aciclic Graph*). O AIG representa um circuito digital utilizando as portas lógicas And e Inversoras. Os nodos do grafo podem representar as variáveis (nodo terminal) e portas lógicas do tipo And. As suas arestas representam as conexões entre as portas lógicas e, quando indicado, a presença de um inversor na conexão.

Um exemplo de um AIG pode ser visto na figura 4.1. Esse grafo representa a equação Booleana $S(a,b,c) = \overline{((a \cdot \overline{b}) \cdot \overline{(\overline{a} \cdot c)}) \cdot (\overline{a} \cdot c)}$. Note que a equação tem 5 negações, pois a equação tem 5 barras indicando inversões. Porém o grafo do AIG tem apenas 4 arestas pontilhadas indicando negações/inversões. Esta diferença acontece porque o nodo $\overline{a} \cdot c$ tem fanout dois e é repetido duas vezes na equação.

Figura 4.1: AIG - *And-Inverter Graph*

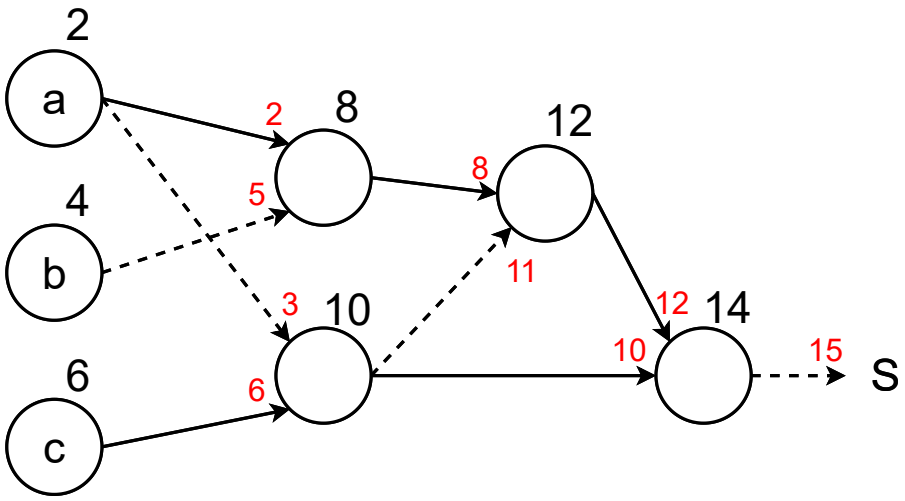


Fonte: Autor e Orientador

4.2 Uso de números pares e ímpares como negação

A estrutura de dados dos AIGs usa números inteiros para representar os identificadores dos nodos. É como se cada nodo tivesse um numero de matrícula ou número de cartão para usar como sua identidade. Mais do que isto, todos os números de matrícula devem ser números pares, sendo os números ímpares reservados para inversões de sinais. Isto é melhor discutido nas próximas seções.

Figura 4.2: AIG - Índice dos nodos



Fonte: Autor e Orientador

4.2.1 Variável, Literal Direto, Literal Negado

As identidades dos nodos em um AIG devem ser números inteiro pares. Isto significa que tanto as variáveis primárias como os nodos internos recebem uma numeração par. Números impares são reservados para representar inversões. Considere como exemplo o AIG mostrado na Figura 4.2. Os números 0 e 1 estão reservados para as constantes Booleanas 0 e 1. As variáveis primárias recebem os números pares 2, 4 e 6. Os nodos tipo and, internos ao AIG, recebem também os números pares 8, 10, 12 e 14. O nodo 8 representa a função $a \cdot \overline{b}$. A variável a é representada pelo número par 2, enquanto a variável negada $\overline{b}$ é representada pelo número impar 5. A mesma situação acontece com nodos intermediários como o nodo 10, correspondente a função $\overline{a} \cdot c$, que pode ser consumido em modo direto (inteiro 10) ou negado (inteiro 11). Esta representação com inteiros pares (para variáveis/nodos diretos) e ímpares (para variáveis/nodos negados) facilita a

representação de um AIG como um vetor de inteiros, conforme será descrito na próxima seção.

4.3 AIG como um vetor

Um AIG pode ser armazenado como um vetor de inteiros, e esta é de fato a melhor estrutura de dados. Esta implementação como vetor de inteiros aumenta consideravelmente a velocidade e facilidade da manipulação do AIG. As características de um AIG que justificam a implementação como vetor de inteiros são discutidas a seguir.

Identificadores inteiros. O fato de os identificadores dos nodos serem inteiros favorece o uso de um vetor como estrutura de dados. Isto acontece porque os próprios identificadores servem de índices (do vetor) para buscar as informações do nodo. Com o uso de pares e ímpares para representar nodos e sua inversão, respectivamente, podemos usar o identificador dividido por dois como índice do vetor.

Tamanho fixo. Por possui uma quantidade fixa de memória por nodo, o AIG nós dá a capacidade de implementa-lo como um vetor de inteiros, pois o tamanho fixo de dois inteiros por nodo permite o uso de um vetor.

Ordem topológica. Outro aspecto que consideramos é a ordem topológica dos nodos no vetor, isso faz com que a complexidade da travessia na estrutura seja linear, colaborando na velocidade de manipulação.

Figura 4.3: AIG - Descrito como um vetor

Index	0	1	2	3	4	5	6	7
Big	0	2	4	6	5	6	11	12
Small	0	2	4	6	2	3	8	10

Fonte: Autor e Orientador

Utilizando como exemplo o AIG da figura 4.2, iremos converte-lo para o vetor ilustrado na figura 4.3. O índice zero corresponde as constantes 0 e 1. Os índices 1, 2, e 3 correspondem as entradas primárias 2, 4 e 6, respectivamente. Entradas primárias são sinalizadas armazenando duas vezes o valor do próprio índice nos vetores big e $small$. O índice 4 no vetor corresponde ao nodo 8. Note que no grafo o nodo 8 tem como entrada os sinais 2 e 5 (4 negado), conforme ilustrado na Figura 4.2. Os índices 5, 6 e 7 representam

os nodos and 10, 12 e 14; respectivamente. A convenção usada é a mesma usada para o nodo 8 (índice 4), conforme pode ser verificado por comparação com a figura 4.2.

4.4 Por que um AIG é semicanônico?

Um AIG é semicanônico porque não podem existir dois nodos diferentes com o mesmo par de entradas. Ou seja, ao se criar duas $ands$ diferentes com entradas 17 e 32 estas duas $ands$ seriam unificadas no mesmo nodo. Porém mesmo assim podem existir nodos distintos no AIG com a mesma função lógica.

4.5 Contribuição deste capítulo

Este capítulo apresentou os AIGs como um tipo de estrutura de dados semicanônica para representação de funções Booleanas. Foi feita a apresentação de AIGs como vetores de inteiros, o que é parte essencial para a eficiência da estrutura de dados. Um fato a reter deste capítulo é que como funções Booleanas são representadas em AIGs como números inteiros, este inteiro poderia ser visto como um número de matrícula da função Booleana na estrutura de dados (do tipo AIG).

5 REVISÃO: BDDS COMO ESTRUTURA CANÔNICA

Neste capítulo apresentamos os diagramas de decisão binários (BDDs, *Binary Decision Diagrams*, em inglês) como um tipo de estrutura de dados canônica. A revisão bibliográfica tenta cobrir os principais aspectos de BDDs de maneira didática.

5.1 Árvores de decisão binárias

BDDs são usados para representar funções Booleanas. Um exemplo de função Booleana é mostrado na tabela verdade da figura 5.1.

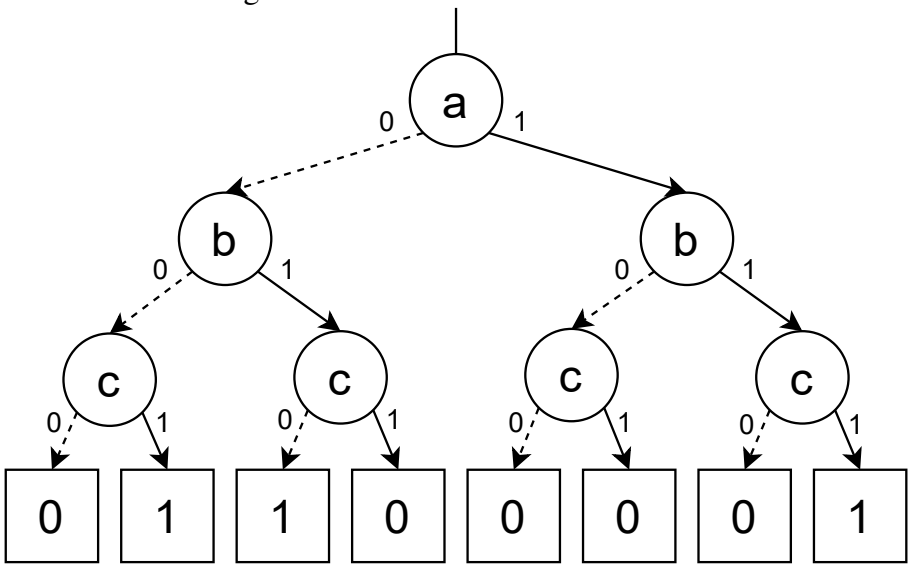
Figura 5.1: Tabela verdade de uma função exemplo

A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Fonte: Autor e Orientador

A tabela verdade da figura 5.1 pode ser reescrita na forma de uma árvore de decisão binária, conforme mostrado na figura 5.2. A árvore de decisão binária tem correspondência direta com a tabela verdade. Os nodos quadrados são nodos terminais, correspondentes a saída da função. Assim o nodo terminal mais a esquerda corresponde a linha 000 (entrada) da tabela verdade e tem o valor 0 (saída) da tabela verdade. De modo similar, o nodo terminal mais a direita corresponde a linha 111 (entrada) da tabela verdade e tem o valor 1 (saída) da tabela verdade. Note que a tabela verdade lida de cima para baixo apresenta a sequência de valores de saída 01100001, que é a mesma sequência dos nodos terminais da árvore binária lida da esquerda para a direita.

Figura 5.2: Árvore de decisão binária



Fonte: Autor e Orientador

5.2 Regras de Redução

A árvore de decisão binária pode ser reduzida para um BDD aplicando-se duas regras de redução. Estas regras são apresentadas nas próximas duas sub-seções.

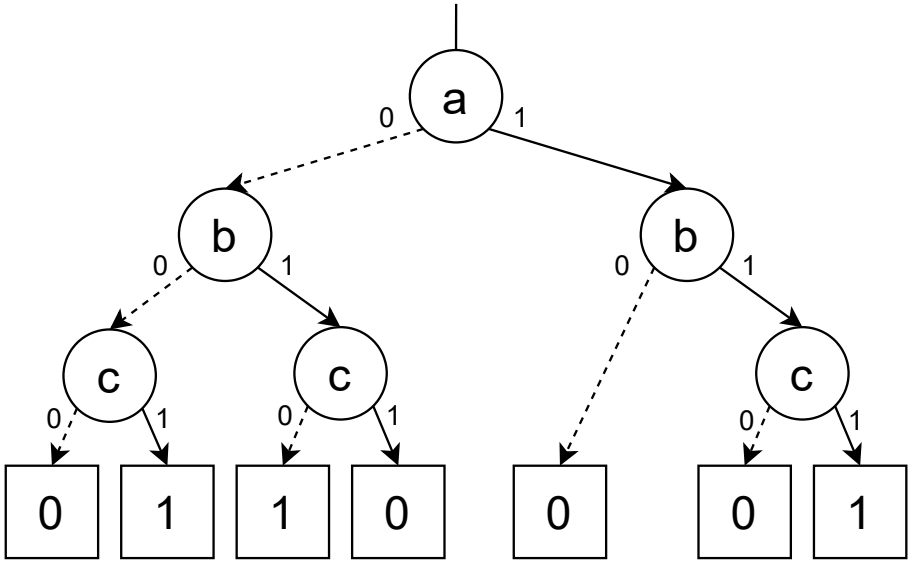
5.2.1 Regra 1: dois filhos iguais

A primeira regra de redução diz que um nodo de BDD que aponta para dois filhos iguais pode ser eliminado já que este nodo não toma decisões. Considere a arvore binária mostrada na na figura 5.2, esta árvore tem um nodo controlado pela variável c que aponta para dois nodos terminais com a constante 0. Assim, o nodo controlado pela variável c terá como resposta sempre o valor 0, independente do valor da variável c. Assim este nodo pode ser eliminado, com o nodo acima dele (controlado pela variável b) apontando diretamente para o nodo terminal 0. A simplificação resultante é mostrada na figura 5.3.

5.2.2 Regra 2: partilhamento de nodos repetidos

A segunda regra de redução diz que dois nodos distintos de um BDD podem ser partilhados (em uma cópia única, ao invés de duas), desde que 1) tenham a mesma variável de controle; 2) apontem para o mesmo filho 0; e 3) apontem para o mesmo filho 1.

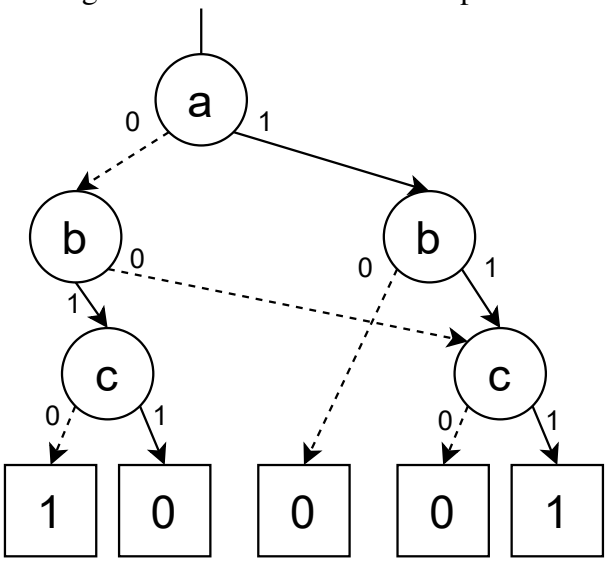
Figura 5.3: Reduzindo nodo com filhos iguais



Fonte: Autor e Orientador

Considere a arvore binária mostrada na na figura 5.3. Esta árvore tem dois nodos distintos controlados pela variável c, onde o filho 0 aponta para o nodo terminal 0 e o filho 1 aponta para o nodo terminal 1. São os dois nodos controlados pala variável c que estão mais a direita e mais a esquerda na árvore da figura 5.3 que se encontram nesta situação. Assim, estes dois nodos podem ser agrupados em uma cópia única. A simplificação resultante é mostrada na figura 5.4.

Figura 5.4: Reduzindo nodos repetidos

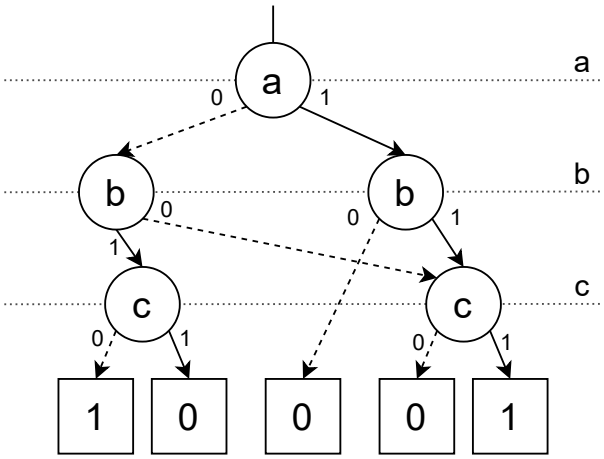


Fonte: Autor e Orientador

5.3 Ordenamento

BDDs também pode ser ordenados, significando que as variáveis devem aparecer em uma ordem ou nível específico de leitura. Os BDDs aqui apresentados já foram construídos com a ordem abc em níveis bem específicos, com a variável a próxima a raiz da árvore, a variável b sendo a variável intermediária, e a variável c próxima aos nodos terminais. A figura 5.5 ilustra este conceito de ordem em BDDs.

Figura 5.5: Árvore com linhas pontilhadas mostrando os níveis das variáveis e a ordem do BDD.



Fonte: Autor e Orientador

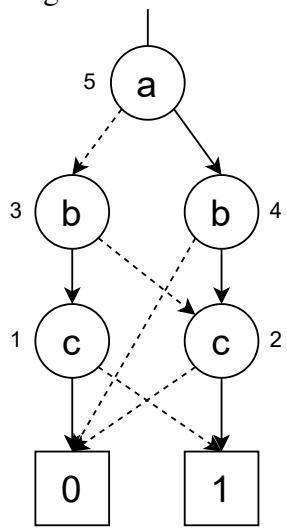
5.4 ROBDDs

Um tipo especial de BDDs são os BDDs reduzidos e ordenados, ou ROBDDs. Nestes ROBDDs, as regras de redução e ordenamento descritas anteriormente se aplicam. ROBDDs são canônicos, ou seja: em um ROBDD dois nodos distintos sempre representam funções Booleanas distintas. Um exemplo de ROBDD é apresentado na figura 5.6.

5.4.1 Strong canonical form

Na forma fortemente canônica de um ROBDD não há necessidade de fazer as simplificações listadas anteriormente. Isto acontece porque o ROBDD já é construído de forma simplificada, com o auxílio de chaves únicas conforme explicado na próxima

Figura 5.6: ROBDD



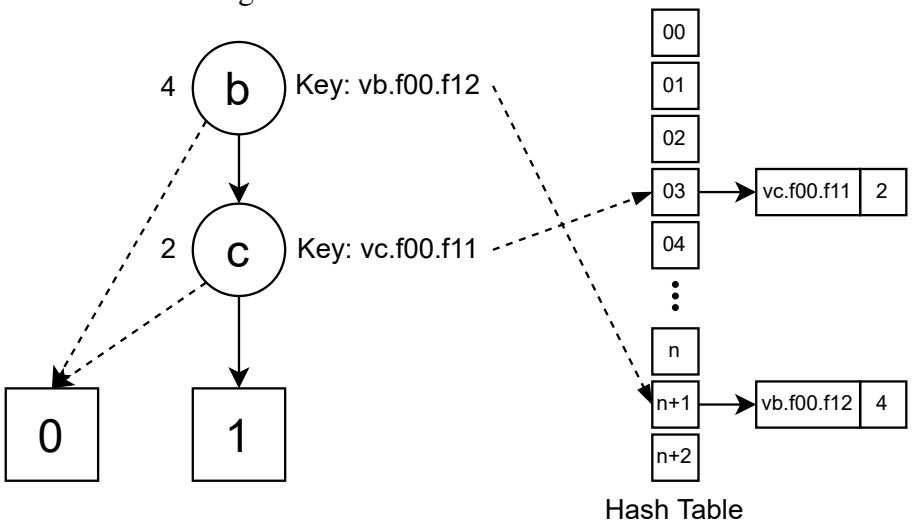
Fonte: Autor e Orientador

seção.

5.4.2 Chave única e hash table

É possível utilizar uma hash table (também conhecida como hashmap) para obrigar a construção do BDD diretamente na sua forma reduzida e ordenada (ROBDD). Utilizaremos para a explicação da hash table e da chave única parte do ROBDD da figura 5.6, contendo apenas os nodos com identificadores 2 e 4, repetidos na figura 5.7, para facilitar o entendimento.

Figura 5.7: Chave única e a hash table



Fonte: Autor e Orientador

A estrutura hash table nos permite guardar uma informação utilizando uma chave. No caso do BDD, a chave utilizada será uma composição da variável do nodo e os seus nodos filhos, que é única para uma função Booleana no formato ROBDD. Por exemplo no nodo 4, temos a chave sendo vb.f00.f12, vb significa que essa chave é de um nodo com a variável b; f00 significa que o filho 0 é igual ao nodo terminal 0 e f12 significa que o filho 1 é igual ao nodo 2. De modo similar, o nodo 2 possuí a chave única igual a vc.f00.f11. Note que está chave será de fato única, nunca existindo dois nodos diferentes possuindo a mesma chave.

Utilizando a chave única de cada nodo iremos guardar na hash table o identificador do nodo que possuí aquela chave. Podemos ver na figura 5.7 que o nodo 2 guarda seu identificador na posição 3 da hash table e o nodo 4 guarda seu identificador na posição n+1. É omitido a função de hash neste exemplo.

A hash table força a construção do ROBDD pois ela é sempre consultada, com complexidade constante, antes da criação de um novo nodo evitando assim repetição de nodos e mantendo a canonicidade do BDD.

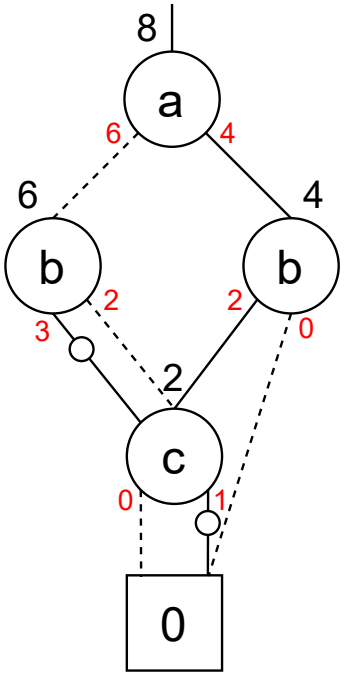
5.5 BDDs com arcos negados

BDDs podem utilizar arcos negados de forma que uma função e seu inverso estejam codificadas no mesmo nodo do ROBDD. Um exemplo de ROBDD com arcos negados é apresentado na figura 5.8. Este ROBDD corresponde a mesma função apresentada no ROBDD da figura 5.6, sem o uso de arcos negados. A numeração dos nodos na figura 5.8 é explicada a seguir.

5.5.1 Uso de numeração compatível com AIGs

O ROBDD da figura 5.8 tenta usar um esquema de numeração que é similar ao do AIGs. O nodo com a variável c tem a numeração 2, e o inteiro 3 é usado para representar sua negação. As negações são representadas nos arcos através de um circulo (bolinha), pois a notação de linha continua e tracejada é usada para identificar qual dos arcos é o filho 0 (linha tracejada) e qual dos arcos é o filho 1 (linha contínua).

Figura 5.8: ROBDD com arcos negados



Fonte: Autor e Orientador

5.6 Contribuição deste capítulo

Este capítulo apresentou os BDDs como um tipo de estrutura de dados canônica para representação de funções Booleanas. Foi feita a apresentação de BDDs incorporando algumas características dos AIGs, para aumento da eficiência da estrutura de dados. Um fato a reter deste capítulo é que como funções Booleanas são representadas em BDDs como números inteiros, este inteiro poderia ser visto como um número de matrícula da função Booleana na estrutura de dados (do tipo BDD).

6 PROPOSTA TCC: INVESTIGAR UNIFICAÇÃO ENTRE AIGS E BDDS

Considerando a intenção de criar uma estrutura mista que agregue as características de AIGs e de ROBDDs, começando por tentar usar uma numeração compatível entre uma cópia de um AIG e uma cópia de um ROBDD dentro de uma mesma classe. Porém alguns problemas surgem ao tentar usar esta unificação. Este capítulo tenta fazer uma comparação entre BDDs e AIGs em termos de processo de numeração de nodos. É necessário ter consciência destas diferenças para propor uma estrutura de dados híbrida que seja eficiente.

6.1 Hipótese inicial: unificação entre AIGs e BDDs

A hipótese inicial deste TCC partiu de observações a respeito dos AIGs e dos BDDs enquanto estrutura de dados. Estas observações são listadas a seguir.

Observação 1: Os algoritmos de composição funcional usam um par {funcionalidade, estrutura} que pode ser representado como um par do tipo {BDD, AIG}.

Observação 2: Uma função em um AIG pode ser representada por um inteiro.

Observação 3: Uma função em um BDD pode ser representada por um inteiro.

Destas três observações surge naturalmente a pergunta investigada neste TCC. A pergunta é formulada a seguir.

Pergunta investigada no TCC: Considerando se o conjunto das três observações, o par {funcionalidade, estrutura} tipo {BDD, AIG} é na verdade um par do tipo {inteiro, inteiro}. Assim a pergunta natural que se faz é se este par não poderia ser substituído por um único inteiro (ao invés de um par de inteiros), forçando as duas estruturas a manterem a mesma numeração sequencial durante a construção das funções Booleanas.

6.2 Número de referência par ou impar

Apesar de os números de referência dos nodos de AIG e dos nodos de um BDD serem pares a criação da função Booleana representada pelo nodo de referência (par) de um AIG pode resultar no cadastro desta função como um número impar no BDD. Assim, há necessidade de tradução de números. Ou seja: na prática ao criar o nodo 42 do AIG em um BDD, este pode receber o número 43 no BDD (ou 42).

6.3 Numeração sequencial

Ao se criar um nodo em um AIG, não há a criação de nodos intermediários na estrutura de dados. Porém ao se criar um novo nodo em um ROBDD , vários nodos intermediários novos podem surgir, criando novos números. Ou seja: na prática ao criar o nodo 42 do AIG em um BDD, este pode receber o número 48 no BDD (nodos 44 e 46 criados internamente).

6.4 Falta de canonicidade no AIG

O AIG não tem canonicidade, assim vários nodos do AIG podem ter o mesmo número no BDD. Ou seja: na prática ao criar os nodos 42, 54 e 86 do AIG em um BDD, estes três nodos podem receber o número 48 no BDD (BDD retorna o mesmo nodo por ser canônico e os três nodos do AIG representarem a mesma função Booleana). Notem que isto pode ser considerado na estrutura do AIG, com a inserção de nodos de escolha para diferentes implementações da mesma função.Assim,seria possível numerar apenas o nodo de escolha no AIG.

6.5 Possibilidade de ciclos no AIG

Parte da eficiência dos algoritmos de AIGs se deve ao fato de o grafo ser dirigido e acíclico e armazenado em ordem topológica em um vetor. Isto faz com que algoritmos de grafos possam ser executados de modo vetorizado sem controle especifico da estrutura do grafo, devido ao armazenamento em ordem topológica. Porém, as diferentes numerações do BDD e do AIG podem fazer com que esta ordem topológica seja perdida.

6.6 Contribuição deste capítulo

Este capítulo apresentou as dificuldades de sincronizar a numeração de nodos de um AIG com a numeração de nodos de um BDD. Apesar de não termos apresentado uma solução, este capítulo é uma contribuição para a sistematização do conhecimento visando pesquisas futuras. Nos seis meses deste TCC não encontramos uma resposta positiva para a questão de numeração única eficiente. Mas a questão permanece em aberto.

7 PROPOSTA: AIG COMPLETAMENTE CANÔNICO

Neste capítulo apresentamos a nova estrutura de dados que une o AIG com o BDD para formar um par funcionalidade-estrutura {BDD, AIG} canônico que chamamos de AIG completamente canônico. Realizamos a descrição conceitual e apresentamos partes da implementação desta estrutura de dados.

7.1 Descrição

O objetivo desta proposta é fazer uma implementação do algoritmo de composição funcional voltado para o par {BDD, AIG}, ao invés de ser voltado para o par original {Tabela Verdade, Equação}. Desta forma implementamos um pacote de AIGs e outro pacote de ROBDDs com arco negado, estruturas descritas nos capítulos anteriores, e passamos a considerar o identificador de cada nodo em cada estrutura sendo o mesmo identificador.

Foi também construído o conjunto de métodos para realizar a composição funcional descritos no artigo original (REIS et al., 2009). Esses métodos consistem na manipulação do par proposto {BDD, AIG} em uma série de baldes (buckets) realizando a combinação dos pares e distribuindo cada par em baldes levando a quantidades de nodos que o AIG possuí.

A figura 7.1 mostra uma visão da arquitetura de software desenvolvida. Na próxima seção explicaremos em mais detalhes cada uns dos elementos .

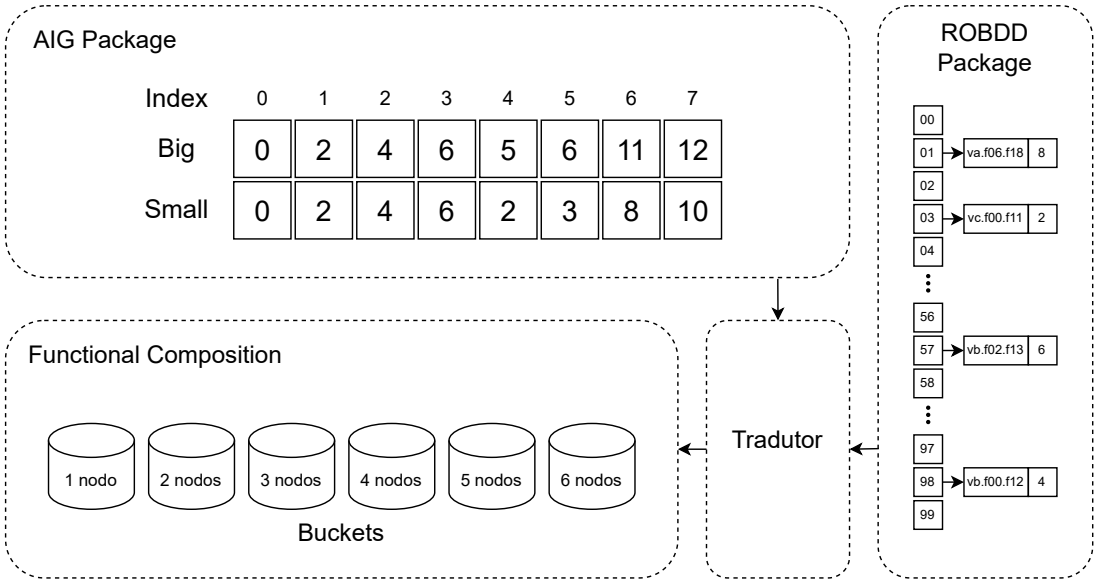
7.2 Estrutura de Dados

A arquitetura principal do software desenvolvido está distribuído em 4 elementos. São eles o Pacote AIG, o Pacote ROBDD, o Pacote de composição funcional e o Tradutor. Nessa seção Explicaremos em mais detalhes cada um dos elementos propostos para a criação da composição funcional utilizando um par {BDD, AIG}.

7.2.1 Pacote AIG

O pacote AIG consistem primeiramente dos vetores big e $small$ que armazenam a estrutura do grafo de uma forma muito eficiente. Junto neste pacote estão métodos

Figura 7.1: Estrutura de dados utilizada



Fonte: Autor e Orientador

para a manipulação dos nodos do AIG. Métodos para criar ou remover nodos, realizar combinações do tipo AND e OR com os nodos e etc.

7.2.2 Pacote ROBDD

O pacote ROBDD possui uma implementação do ROBDD com arcos negados para que pudéssemos igualar os identificadores das duas estruturas. A principal estrutura de dados deste pacote é a hash table. Junto neste pacote estão métodos mais complexos que permitem a criação do ROBDD com arcos negados. Um desses métodos é o ITE(If-Then-Else) muito conhecido na literatura para a criação eficiente de nodos de BDD. Outros métodos de manipulação simulares ao do pacote AIG também foram criados.

7.2.3 Pacote Composição Funcional

O pacote de composição funcional desenvolvido consiste na sequência de baldes (buckets) que são utilizado na para distribuir as soluções encontradas considerando um determinado custo, neste caso, a quantidade de nodos no AIG. É neste pacote que se encontra os métodos de permutação que realizam a composição buttom up do par {BDD,

AIG} até ser encontrado um par com a funcionalidade esperada. Note que a composição funcional possui um conjunto solução "infinito"e é preciso de boas eurísticas no método de combinação para que o algoritmo não se perca no mar de funções.

7.2.4 Tradutor

Como neste momento não foi possível a criação de um par {BDD, AIG} ideal, ou seja, auto contido em apenas uma estrutura única e que não necessita de conversão de BDD para AIG (ou vice e versa), se faz necessário a criação de um tradutor entre as duas estruturas. Como descrito no capítulo 6, não é trivial a união do AIG com o ROBDD apenas igualando os seus identificadores, uma serie de considerações foram feitas e sugerem então um tradutor entre as estruturas.

7.3 Contribuição deste capítulo

Este capítulo apresentou uma nova estrutura de dados, o par {BDD, AIG} para ser utilizada junto com a composição funcional. Ainda que partes importantes não tenham sido desenvolvidas devido a alta complexidade, resultados preliminares muito interessantes surgiram a partir desta abordagem, como veremos no capítulo a seguir.

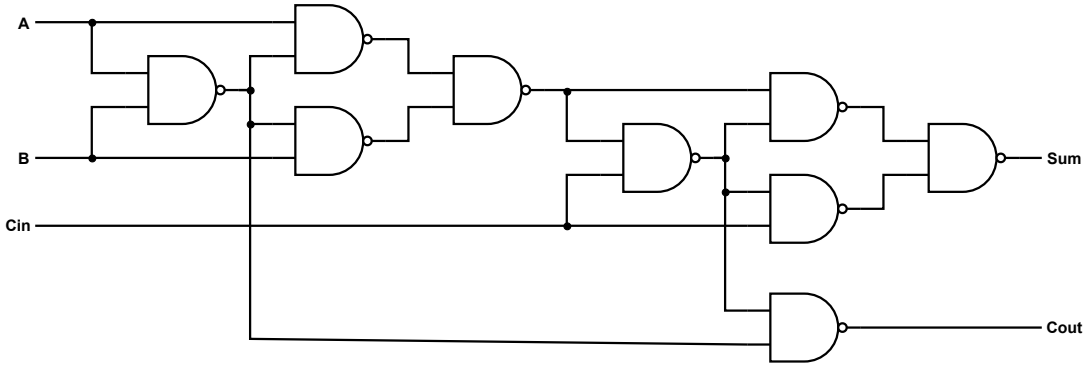
8 RESULTADOS PRELIMINARES

Neste capítulo são apresentados resultados preliminares. Apesar de não termos chegado a uma implementação eficiente para circuitos grandes, chegamos a alguns resultados interessantes para o circuito somador completo.

8.1 Menor circuito conhecido: 9 portas

Em termos de implementação com portas simples de duas entradas, o menor circuito conhecido pelo autor e seu orientador tem nove portas lógicas. Este circuito é apresentado na figura 8.1 em termos de portas NAND de duas entradas. O mesmo circuito é mostrado na figura 8.2 em termos de nodos de AIG. Este circuito servirá de base de comparação para os circuitos descobertos no contexto deste trabalho.

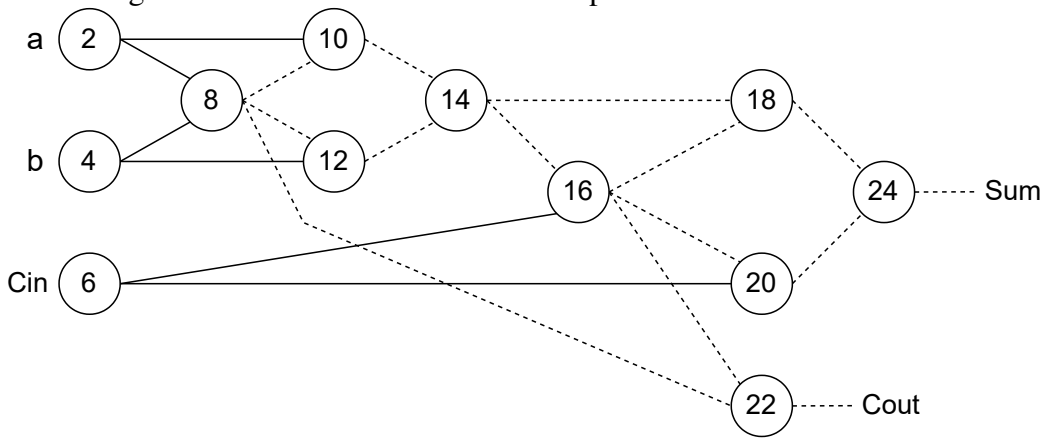
Figura 8.1: Menor circuito conhecido pelos autores: Formato células NAND



Fonte: Autor e Orientador

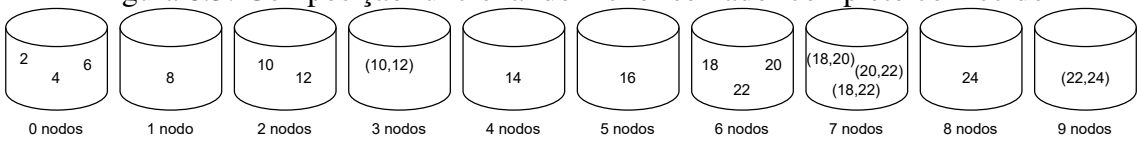
Na figura 8.3 apresentamos o que seria a composição funcional para encontrar o circuito na figura 8.2. Esta não é derivada de uma execução completamente automática do código, mas sim uma idealização do que um algoritmo ideal (a ser buscado) produziria. A figura tem a intenção de deixar mais claro ao leitor como reproduzir os métodos, estruturas de dados e ideias discutidos neste trabalho, de modo a permitir a reprodução e a continuação deste estudo.

Figura 8.2: Menor circuito conhecido pelos autores: Formato AIG



Fonte: Autor e Orientador

Figura 8.3: Composição funcional do menor somador completo conhecido

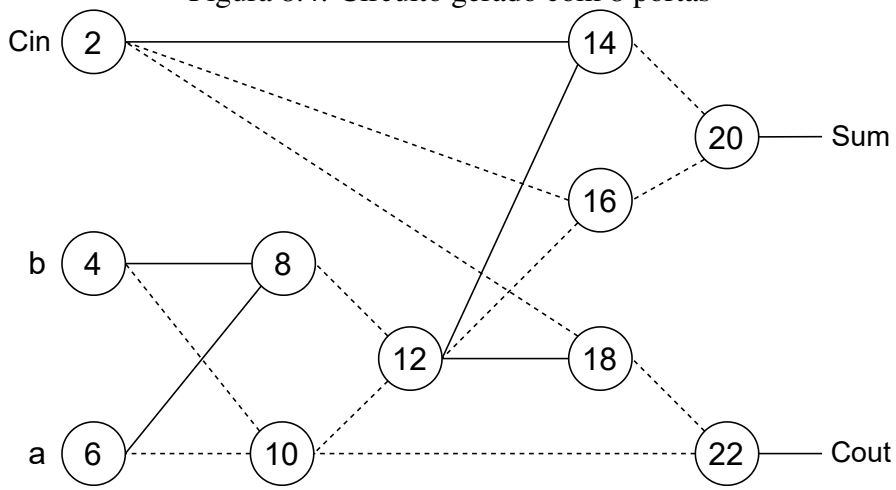


Fonte: Autor e Orientador

8.2 Uma primeira contribuição: Circuito com 8 portas

Um primeiro circuito descoberto nesta pesquisa tem 8 portas lógicas para implementar um somador completo. Este circuito é mostrado na figura 8.4.

Figura 8.4: Circuito gerado com 8 portas

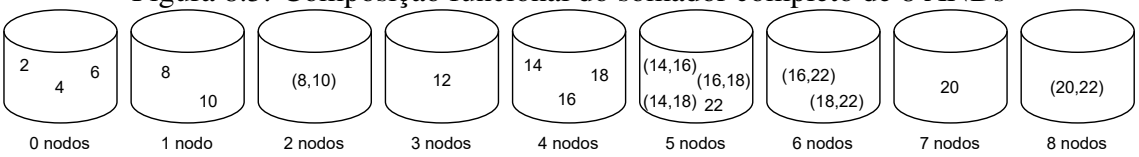


Fonte: Autor e Orientador

Na figura 8.5 apresentamos o que seria a composição funcional para encontrar o circuito na figura 8.4. De novo, a figura tenta esclarecer ao leitor como reproduzir os métodos, estruturas de dados e ideias discutidos neste trabalho, de modo a permitir a

reprodução e a continuação deste estudo.

Figura 8.5: Composição funcional do somador completo de 8 ANDs

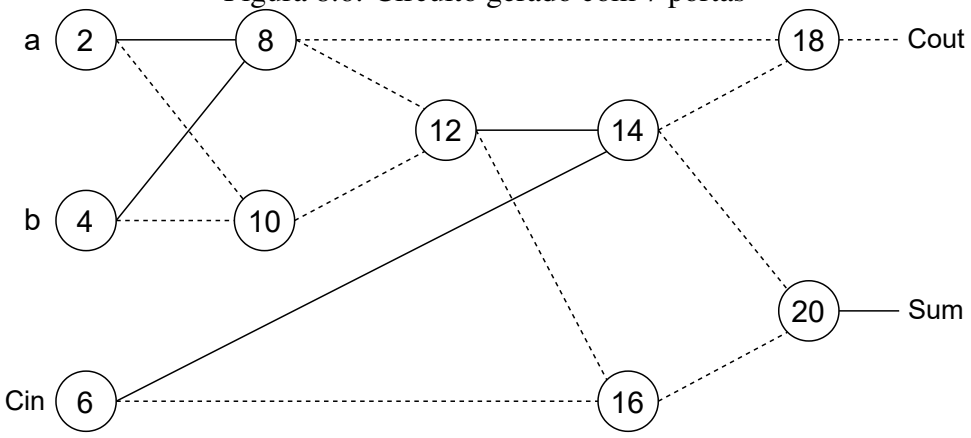


Fonte: Autor e Orientador

8.3 Uma segunda contribuição: Circuito com 7 portas

Um segundo circuito descoberto nesta pesquisa tem 7 portas lógicas para implementar um somador completo. Este circuito é mostrado na figura 8.6

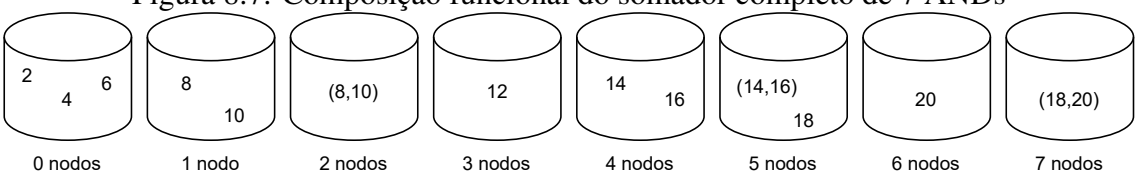
Figura 8.6: Circuito gerado com 7 portas



Fonte: Autor e Orientador

Na figura 8.7 apresentamos o que seria a composição funcional para encontrar o circuito na figura 8.6. Mais uma vez, ressaltamos que a figura serve para ajudar na reprodução e na continuação deste estudo.

Figura 8.7: Composição funcional do somador completo de 7 ANDs



Fonte: Autor e Orientador

8.4 Análise dos resultados

Os resultados preliminares obtidos para o somador completo demonstraram uma interessante redução de 9 para 7 portas lógicas, sem contar os inversores, mantendo o nível lógico crítico da entrada Cin para a saída $Cout$ em 2 níveis e reduzindo o fanout máximo de 3 para 2.

O somador completo mínimo composto por células lógicas conhecido pelo autor e orientador possui 9 portas NAND. Portanto, os resultados sugerem uma metodologia de síntese promissora que vale a pena ser investigada com mais profundidade em trabalhos futuros.

8.5 Contribuição deste capítulo

Este capítulo apresentou resultados preliminares deste estudo. Pode se destacar que foram apresentadas duas versões de somadores completos com um número menor de portas comparado com a versão conhecida com o menor número de portas simples.

9 CONCLUSÃO

Este trabalho de conclusão apresentou uma proposta inicial para uma nova estrutura de dados que integre as funcionalidades de um pacote de BDDs com um pacote de AIGs. Esta integração se revelou mais complexa do que o esperado, então ainda não chegamos em uma solução eficiente. Porém as tentativas de gerar resultados criaram resultados preliminares interessantes. Dentre estes podemos destacar duas contribuições principais.

Em primeiro lugar o comparativo apresentado no capítulo 6 representa um avanço no conhecimento que será importante para criar uma estrutura de dados mais efetiva. Apesar de não ter sido encontrada uma solução para o uso de numeração sequencial única, o capítulo 6 é uma contribuição para a sistematização do conhecimento visando pesquisas futuras. Nos seis meses deste TCC não encontramos uma resposta positiva para a questão de numeração única eficiente. Mas a questão permanece em aberto.

Em segundo lugar, uma versão inicial do algoritmo de composição funcional usando a estrutura de dados (ainda não estável) implementada na proposta foi capaz de gerar somadores completos com 7 e 8 portas lógicas de duas entradas. Pelo melhor conhecimento do autor e seu orientador, o menor circuito conhecido tem nove portas. Note que esta contribuição deve ser considerada com alguma precaução, por três fatores. Em primeiro lugar, talvez circuitos de oito e sete portas lógicas já sejam conhecidos, apesar do desconhecimento do autor e seu orientador. Em segundo lugar, o circuito de nove portas conhecido pode ser feito com nove portas nand2, portanto sem usar inversores extras. Em terceiro lugar, circuitos usando portas complexas com um número bem menor de transistores são conhecidos.

Para terminar em uma nota positiva, os avanços aqui apresentados podem resultar em pesquisas promissoras, desde que mais aprofundados. Por fim, consideramos que as contribuições apresentadas são suficientes para um TCC e podem ser aprofundadas em trabalhos futuros.

REFERÊNCIAS

BUTZEN, P. et al. Design of cmos logic gates with enhanced robustness against aging degradation. **Microelectronics Reliability**, v. 52, n. 9, p. 1822–1826, 2012. ISSN 0026-2714. SPECIAL ISSUE 23rd EUROPEAN SYMPOSIUM ON THE RELIABILITY OF ELECTRON DEVICES, FAILURE PHYSICS AND ANALYSIS. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0026271412002892>.

BUTZEN, P. F. et al. Transistor network restructuring against nbti degradation. **Microelectronics Reliability**, v. 50, n. 9, p. 1298–1303, 2010. ISSN 0026-2714. 21st European Symposium on the Reliability of Electron Devices, Failure Physics and Analysis. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0026271410004130>.

BUTZEN, P. F. et al. Standby power consumption estimation by interacting leakage current mechanisms in nanoscaled cmos digital circuits. **Microelectronics Journal**, Elsevier, v. 41, n. 4, p. 247–255, 2010.

GOMES, I. et al. Using only redundant modules with approximate logic to reduce drastically area overhead in tmr. In: IEEE. **Test Symposium (LATS), 2015 16th Latin-American**. [S.l.], 2015. p. 1–6.

GOMES, I. A. et al. Methodology for achieving best trade-off of area and fault masking coverage in atmr. In: IEEE. **Test Workshop-LATW, 2014 15th Latin American**. [S.l.], 2014. p. 1–6.

JUNIOR, L. S. da R. et al. Fast disjoint transistor networks from bdds. In: ACM. **Proceedings of the 19th annual symposium on Integrated circuits and systems design**. [S.l.], 2006. p. 137–142.

MACHADO, L. et al. Kl-cut based digital circuit remapping. In: IEEE. **NORCHIP, 2012**. [S.l.], 2012. p. 1–4.

MARRANGHELLO, F. S. et al. Factored forms for memristive material implication stateful logic. **IEEE Journal on Emerging and Selected Topics in Circuits and Systems**, IEEE, v. 5, n. 2, p. 267–278, 2015.

MARTINS, M. G.; RIBAS, R. P.; REIS, A. I. Functional composition: A new paradigm for performing logic synthesis. In: IEEE. **Quality Electronic Design (ISQED), 2012 13th International Symposium on**. [S.l.], 2012. p. 236–242.

MARTINS, M. G. A. et al. Boolean factoring with multi-objective goals. In: **2010 IEEE International Conference on Computer Design**. [S.l.: s.n.], 2010. p. 229–234.

MISHCHENKO, A. **Towards Next Generation Logic Synthesis and Verification, disponível em https://www.ufrgs.br/chip-in-the-minuano/**. 2022.

MOREIRA, M. et al. Semi-custom ncl design with commercial eda frameworks: Is it possible? In: **International Symposium on Asynchronous Circuits and Systems (ASYNC), Potsdam**. [S.l.: s.n.], 2014.

NEUTZLING, A. et al. Synthesis of threshold logic gates to nanoelectronics. In: IEEE. **2013 26th Symposium on Integrated Circuits and Systems Design (SBCCI)**. [S.l.], 2013. p. 1–6.

NEUTZLING, A. et al. Threshold logic synthesis based on cut pruning. In: IEEE. **2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.], 2015. p. 494–499.

POLI, R. E. et al. Unified theory to build cell-level transistor networks from bdds [logic synthesis]. In: IEEE. **16th Symposium on Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings.** [S.l.], 2003. p. 199–204.

REIS, A. et al. Fast boolean factoring with multi-objective goals. In: **International Workshop on Logic and Synthesis - IWLS 2009**. [S.l.: s.n.], 2009.

REIS, A. I. Covering strategies for library free technology mapping. In: IEEE. **Proceedings. XII Symposium on Integrated Circuits and Systems Design (Cat. No. PR00387)**. [S.l.], 1999. p. 180–183.

REIS, A. I.; DRECHSLER, R. **Advanced logic synthesis**. [S.l.]: Springer, 2018.

ROSA, L. et al. Scheduling policy costs on a java microcontroller. In: SPRINGER. **On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops**. [S.l.], 2003. p. 520–533.

ROSA, L. S. da et al. A comparative study of cmos gates with minimum transistor stacks. In: **Proceedings of the 20th annual conference on Integrated circuits and systems design**. [S.l.: s.n.], 2007. p. 93–98.

ROSA, L. S. da et al. Switch level optimization of digital cmos gate networks. In: IEEE. **2009 10th International Symposium on Quality Electronic Design**. [S.l.], 2009. p. 324–329.

SILVA, D. N. da; REIS, A. I.; RIBAS, R. P. Cmos logic gate performance variability related to transistor network arrangements. **Microelectronics Reliability**, Pergamon, v. 49, n. 9-11, p. 977–981, 2009.

TOGNI, J. et al. Automatic generation of digital cell libraries. In: IEEE. **Proceedings. 15th Symposium on Integrated Circuits and Systems Design**. [S.l.], 2002. p. 265–270.

WAGNER, F.; REIS, A.; RIBAS, R. Fundamentos de circuitos digitais. **Sagra Luzzatto, Porto Alegre**, 2006.