# Utilização de Diagramas de Decisão Binária (BDDs) para Representar Máquinas de Estado (FSMs) e Verificar Fórmulas de Lógica Temporal (CTL)

*Paulo Flores*

INESC/IST
20 Julho 1996

**Resumo**

Neste relatório descreve-se sumariamente a representação de funções booleanas utilizando diagramas de decisão binária (BDDs - *Binary Decision Diagrams*). A forma como as BDDs podem ser usadas para representar/enumerar todos os estados acessíveis de uma máquina de estados (FSM - *Finite State Machine*) é descrito com algum detalhe. Baseando-se nesta técnicas são descritos algoritmos para verificação de fórmulas de lógica temporal (CTL - *Computation Tree Logic*) sobre as máquinas de estados.

# Índice

# 1 Introdução

Os algoritmos de travessia de grafos de transições de uma máquina de estados têm aplicações nas áreas de síntese, teste e verificação. Tradicionalmente os métodos para enumerar o espaço de estados de uma máquina de estados tendem a explodir exponencialmente com o número de entradas.

Em [Coudert 89b] foi apresentado pela primeira vez uma forma compacta de representar um conjunto de estados, independente da sua codificação, através da utilização de BDDs. Foram também descritos algoritmos para determinar os valores do contra-domínio de uma função booleana para um sub-conjunto do domínio. Para este tipo de representação foi ainda descrito um algoritmo para atravessar o grafo de transições de uma máquina de estados de forma idêntica às tradicionais formas de pesquisas realizadas primeiro-em-largura (*breadth-first*). Ou seja, determinando sucessivamente o conjunto dos próximos estados de uma dado conjunto inicial de estados. Neste tipo de algoritmos quer as entradas quer os estados são enumerado implicitamente. Em [Burch 90b] este conceito foi extendido a computação de fórmulas de lógica temporal.

Este tipo de algoritmos são geralmente referidos como algoritmos de travessia simbólica, pois envolvem a realização de operações sobre conjuntos de estados, que são representados simbolicamente por BDDs. A principal operação realizada nestes algoritmos é o calculo do contra-domínio de uma função para um dado subconjunto do domínio.

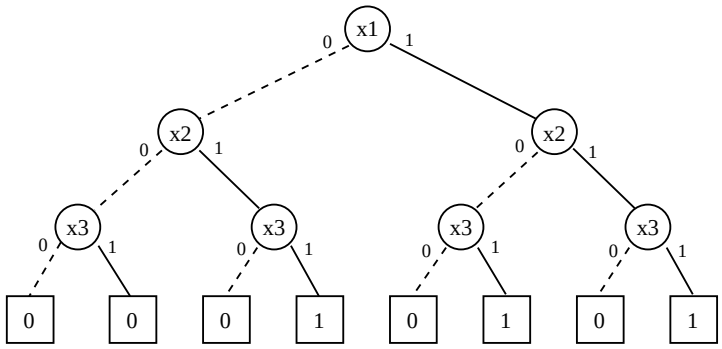
# 2 BDDs - Diagramas de Decisão Binária

## 2.1 Representação de funções booleanas

Existe uma variedade de métodos para representar funções booleanas. As formas tradicionais de representação, tais como tabelas de verdade, mapas de Karnaugh ou somas de produtos canónicas, são computacionalmente impraticáveis, pois a representação de funções de $n$ variáveis requer uma representação de tamanho $2^n$. Outras representações, conjunto de primos e cubos não redundantes, podem também requerer, em certo casos, representações de tamanho exponencial. Estes tipos de representação têm ainda a desvantagem de não terem uma forma canónica de representar uma função, ou seja uma mesma função pode ter várias representações válidas mas diferentes. Isto faz com que os problemas de equivalência de funções ou verificação de tautologias sejam muito difíceis de resolver.

Os diagramas de decisão binários (*Binary Decision Diagrams*-BDDs) foram introduzidos por Lee [Lee 59] e Akers [Akers 78]. Uma BDD representa uma função booleana $f(x_1, x_2, ..., x_n)$, como um grafo unidireccional, sem ciclos e com um nó de entrada (raiz). Nesse grafo um nó não terminal representa uma variável da função, e de cada um desses nós saiem dois arcos correspondentes à respectiva variável tomar o valor 0 ou 1. Os nós terminais do grafo representam um dos valores possíveis para a função, 0 ou 1. Na figura 1 ilustra-se a representação de uma função de três variáveis, $fun(x_1, x_2, x_3)$.

| $x_1$ | $x_2$ | $x_3$ | $fun(x_1, x_2, x_3)$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |




$$fun(x_1, x_2, x_3) = (x_1 + x_2) \cdot x_3$$

Figura 1: Representação de uma função utilizando uma arvore de decisão binária.

Para uma determinar o valor da função resultante de uma atribuição às suas variáveis é realizado um percurso no grafo da função. Este percurso inicia-se no nó de entrada para chegar a um nó terminal, sendo seleccionado em cada nó o arco correspondente ao valor atribuído à variável que esse nó representa. O valor do nó terminal encontrado é o resultado da função para a atribuição feita.

No entanto, este tipo de representação, tal como apresentada, pode ser feito de forma não canónica. Em [Bryant 86] Bryant introduziu restrições na ordenação das variáveis e algoritmos para transformar uma BDD num diagramas de decisão binários de ordem reduzida (*Reduce Order Binary Decision Diagrams*-ROBDDs). Uma ROBDD é definida da mesma forma que uma BDD com as seguintes restrições:

- Os vértices não terminais encontram-se no grafo por uma dada ordem pré-determinada. Ou seja, para uma ordenação das variáveis $x_1, x_2, ..., x_n$ da função $f(x_1, x_2, ..., x_n)$ tem-se que, para qualquer nó $x_u$ e qualquer dos seus filhos não terminais $x_v$, a seguinte relação de ordem, $ordem(x_u) < ordem(x_v)$.

- O grafo é reduzido, ou seja:

  1. para cada nó do grafo $x_u$, os sub-grafos correspondentes aos arcos $x_u = 0$ e $x_u = 1$ não são isomórficos,
  2. não existem dois nós $x_u$ e $x_v$ cujos sub-grafos sejam isomórficos.

Com estas restrições as ROBDDs (que passaremos a designar simplesmente por BDDs) representam uma função de forma canónica. Na figura 2 representa-se a ROBDD da função $fun(x_1, x_2, x_3)$ atrás apresentada.

A forma e o tamanho de uma ROBDD depende da ordenação escolhida para a variáveis. Apesar do tamanho de representação de uma função com ROBDDs poder crescer exponencialmente (no pior dos casos), existem heurísticas e métodos dinâmicos de ordenação das variáveis que permitem manter o tamanho de representação aceitável para a maioria dos casos de interesse das aplicação de CAD (síntese, teste, verificação, etc).

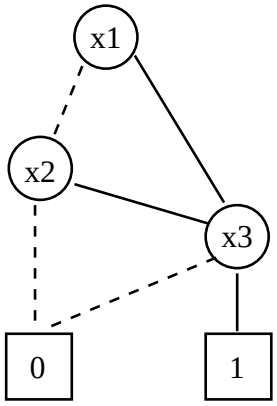



Figura 2: ROBDD da função $fun(x_1, x_2, x_3) = (x_1 + x_2) \cdot x_3$

Recentemente a utilização de BDDs tem-se expandido a áreas de projecto de sistemas concorrentes, lógica matemática e inteligência artificial. Para certos domínios de aplicação a utilização de BDDs requer previamente uma codificação booleana do domínio a representar.

Em muitos domínios de aplicação as BDDs apresentam características próprias que podem ser aproveitadas para reduzir o tamanho de representação e/ou o seu tempo de manipulação. Em [Bryant 95] são descritas treze diferentes formas de representar funções baseadas em BDDs. Uma implementação computacional de um pacote de criação e manipulação de BDDs pode ser encontrado em [Brace 90].

## 2.2 Operações sobre BDDs

Antes de apresentar os algoritmos para representar e atravessar máquinas de estados é necessário definir alguma terminologia que será usada.

A representação de funções sob a forma de BDDs permite de uma forma eficiente realizar algumas manipulações não triviais sobre essas funções booleanas: substituição numa função de todas as instâncias de uma variável por uma outra função e quantificação de uma função por uma variável.

A função que resulta quando algum argumento $x$ de uma função $f$ é restringido a um valor constante $k$ (0 ou 1) é designado de restrição de $f$ ou cofactor de $f$ e representa-se por: $f|_{x \leftarrow k}$. Dado duas restrições a uma função relativamente a uma variável é possível reconstruir uma função da seguinte forma:

$$f = \overline{x} \cdot f|_{x \leftarrow 0} + x \cdot f|_{x \leftarrow 1} \qquad \text{(ou, noutra notação: } f = \overline{x} \cdot f_{\overline{x}} + x \cdot f_x)$$

Esta identidade é geralmente referida como a *expansão de Shannon* da função $f$ em relação a $x$. Note-se que $f|_{x \leftarrow 0}$ e $f|_{x \leftarrow 1}$ correspondem ao dois sub-grafos da BDD que representa $f$, se $x$ for a variável associada ao nó raiz da BDD.

As manipulações não triviais são assim são definidas em termos das funções booleanas básicas e da operação de restrição.

**Definição 1** *A operação de* composição*, onde uma função $g$ substitui uma variável $x$ de uma função $f$, e dado pela seguinte identidade:*

$$f|_{x \leftarrow g} = \overline{g} \cdot f|_{x \leftarrow 0} + g \cdot f|_{x \leftarrow 1}$$

**Definição 2** *A operação de* quantificação existencial *ou operação de* eliminação (smoothing) *de uma variável $x$ sobre uma função $f$ é dada pela seguinte identidade:*

$$\exists x f = f|_{x \leftarrow 0} + f|_{x \leftarrow 1} \qquad \textit{(ou, noutra notação: } S_x f = f_{\overline{x}} + f_x \textit{ )}$$

Se $f : B^n \rightarrow B$ (com $B = \{0, 1\}$) for uma função booleana e $x = (x_1, x_2, ..., x_n)$ um conjunto de variáveis de $f$, então a eliminação em $f$ de $x$ é dado por:

$$S_x f = S_{x_1} ... S_{x_n} f$$

$$S_{x_i} f = f_{\overline{x_i}} + f_{x_i}$$

O operador de eliminação calcula a projecção de $f$ ($f : B^n \rightarrow B$) no sub-espaço $B^n$ ortogonal ao domínio das variáveis de $x$. Várias variáveis podem ser eliminadas numa única passagem através de uma travessia da BDD de baixo para cima.

**Definição 3** *A operação de* quantificação universal *ou operação de* consenso (consensus) *de uma variável $x$ sobre uma função $f$ é dada pela seguinte identidade:*

$$\forall x f = f|_{x \leftarrow 0} \cdot f|_{x \leftarrow 1} \qquad \textit{(ou, noutra notação: } C_x f = f_{\overline{x}} \cdot f_x \textit{ )}$$

Se $f : B^n \rightarrow B$ for uma função booleana e $x = (x_1, x_2, ..., x_n)$ um conjunto de variáveis de $f$, então o consenso em $f$ de $x$ é dado por:

$$C_x f = C_{x_1} ... C_{x_n} f$$

$$C_{x_i} f = f_{\overline{x_i}} \cdot f_{x_i}$$

O operador de consenso pode ser realizado eficientemente utilizando BDDs.

## 2.3 Representação simbólica de conjuntos

**Definição 4** *Uma* função vectorial*, $\vec{f} : B^n \rightarrow B^m$ , é definida como $\vec{f} = [f_1 \cdots f_m]$, sendo $n$ o número de variáveis das funções booleanas $f_1 \cdots f_m$.*

**Definição 5** *A* imagem *de um subconjunto $S \subseteq B^n$ obtido por uma função $\vec{f}$, é o conjunto dos valores dos contra-domínio de $\vec{f}$ quando o domínio desta função é restringido a $S$, ou seja:*

$$\vec{f}(S) = \{y \in B^m : \exists x \in S \wedge y = \vec{f}(x)\}$$

*À imagem do domínio, $\vec{f}(B^n)$, dá-se o nome de* alcance *da função $\vec{f}$.*

*A* imagem inversa *de um subconjunto $T \subseteq B^m$ de uma função $\vec{f}$ é definida da seguinte forma:*

$$\vec{f}^{-1}(T) = \{x \in B^n : \exists y \in T \wedge y = \vec{f}(x)\}$$

**Definição 6** *A* função característica *de um conjunto $S \subseteq B^n$ e a função $\chi_S : B^n \to B$ definida da seguinte forma:*

$$\chi_S(x) = \begin{cases} 1 & x \in S \\ 0 & x \notin S \end{cases}$$

Na figura 3 apresenta-se um exemplo de uma função vectorial, do conjunto representado pelo seu alcance, e da função característica desse sub-conjunto.

| $B^2$ | | $\vec{f}$ | |
|---|---|---|---|
| $x_1$ | $x_2$ | $f_1$ | $f_2$ |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 |

$$\vec{f} = [(x_1 + x_2)(x_1 \oplus x_2)]$$

$A = \{\text{alcance de}\vec{f}\} = \vec{f}(B^2) = \{(0,0),(1,1),(1,0)\}$
$\chi_A = y_1 + \overline{y_2})$

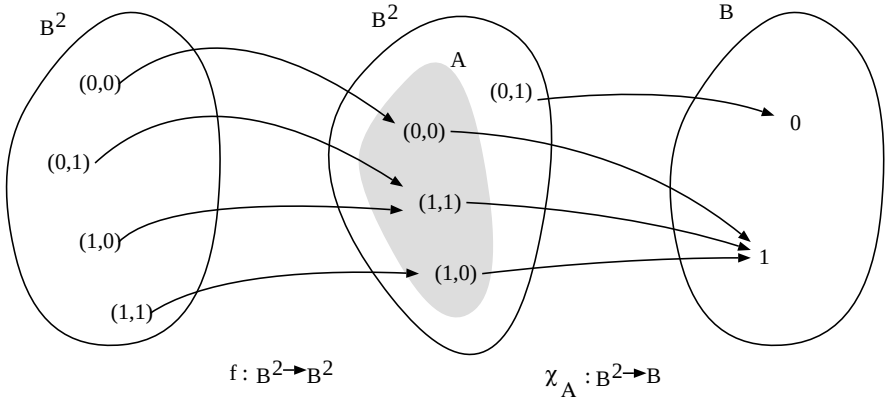



Figura 3: Exemplo de uma função vectorial, do conjunto representado pelo seu alcance, e da função característica desse sub-conjunto.

As funções características são uma representação funcional de conjuntos e subconjuntos, podendo ser definidas para funções ou relações. Se $R$ for uma ralação, $R : A \to Z$, então a função característica de $R \subseteq A \times Z$ é $\chi_R : A \times Z \to B$, definida da seguinte forma:

$$\chi_R(x,y) = \begin{cases} 1 & x \in A, y \in Z \wedge R(x,y) \in R \\ 0 & \text{nos outros casos} \end{cases}$$

A manipulação de conjuntos pode ser feita directamente sobre as funções características que representam esse conjuntos. Assim, se $S$ e $T$ forem dois conjuntos definidos pelas suas funções características, $\chi_S$ e $\chi_t$, temos que:

$$\chi_\emptyset = 0$$

$$\chi_{S \cup T} = \chi_S + \chi_T$$

$$\chi_{S \cap T} = \chi_S \cdot \chi_T$$

$$\chi_{S-T} = \chi_S \cdot \overline{\chi_T}$$

Qualquer conjunto pode ser representado pela sua função características, que por sua vez pode ser eficientemente representado e manipulado utilizando BDDs.

# 3 Máquinas de Estados

## 3.1 Modelo das máquinas de estado

A máquinas de estados são modelos abstractos para descrever o comportamento de circuitos combinatórios, circuitos com memória. Formalmente, uma máquinas de estados determinística, $\mathcal{M}$, é definida como um tuplo de 6 elementos:

$$\mathcal{M} = (I, S, O, \delta, \lambda, \sigma_0)$$

$I$ - representa o conjunto não vazio dos símbolos de entrada da máquina de estados. Para o caso de circuitos digitas, em que o valor de cada entrada $x_i$ pertence ao conjunto $B = \{0, 1\}$, temos que, $I \subseteq B^{ni}$, sendo $ni$ o número de entradas do circuito.

$S$ - representa o conjunto não vazio dos estados possíveis da máquina. Para o caso de circuitos digitas, em que cada estado esta codificado em $b$ bits, temos $S \subseteq B^b$.

$O$ - representa o conjunto não vazio dos símbolos de saída da máquina de estados. Para o caso de circuitos digitas, em que o valor de cada saída $y_i$ pertence ao conjunto $B = \{0, 1\}$, temos que, $I \subseteq B^{no}$, sendo $no$ o número de saídas do circuito.

$\delta$ - é a função que define o próximo estado da máquina, $\delta : I \times S \to S$. Para o caso de circuitos digitas $\vec{\delta} = [\delta_1 \delta_2 \cdots \delta_b]$, em que $\delta_i : S \times I \to B$ representa a função de transição do bit $i$.

$\lambda$ - é a função que define as saídas da máquina, $\lambda : I \times S \to O$ para uma máquina de Mealy, e $\lambda : S \to O$ para uma máquina de Moore. Para o caso de circuitos digitas temos $\vec{\lambda} = [\lambda_1 \lambda_2 \cdots \lambda_{no}]$, em que $\lambda_i : S \times I \to B$ representa a função da saída $y_i$ (máquinas de Mealy).

$\sigma_0$ - representa o estado inicial da máquina, $\sigma_0 \in S$. Para o caso de circuitos digitais $\sigma_0$ é um vector de $b$ bits.

**Definição 7** *Um estado da máquina, $\sigma'$, diz-se* sucessor *(ou* estado seguinte*) de um outro estado, $\sigma$, se e só se: $\exists i \in I : \sigma' = \delta(i, \sigma)$.*

**Definição 8** *Um* sequência de estados *de uma máquina é uma sequência infinita de estados possíveis, designada por $\{\sigma_1, \sigma_2, ..., \sigma_i, ...\}$ e em que $\sigma_i$ é sucessor de $\sigma_{i-1}$.*

**Definição 9** *Um estado $\sigma_f$ de uma máquina é* atingível *se existir uma sequência de estados, $\{\sigma_1, \sigma_2, ..., \sigma_f\}$ em que $\sigma_1 = \sigma_0$.*

## 3.2 Representação de máquinas de estados com BDDs

A representação de máquinas de estados pode ser eficientemente feita utilizando BDDs [Coudert 91, Burch 90a, Bose 89]. Para isso é necessário seleccionar uma codificação binária para os estados da máquina e para o alfabeto de entrada.

A forma mais simples e directa de representar uma máquina de estados utilizando BDDs consiste em representar as $b$ funções de transição, correspondentes às funções transição parciais $y_i = \delta_i(x, s)$, com BDDs.

Outra forma de representar uma máquina de estados é feita à custa de uma função característica $\eta(x, s, s')$, que terá o resultado de 1 quando a entrada $x$ pode originar uma transição do estado $s$ para o estado $s'$. Como exemplo apresenta-se na figura 4 o diagrama de estados de uma máquina e a representação da sua função característica $\eta(x, s, s')$.

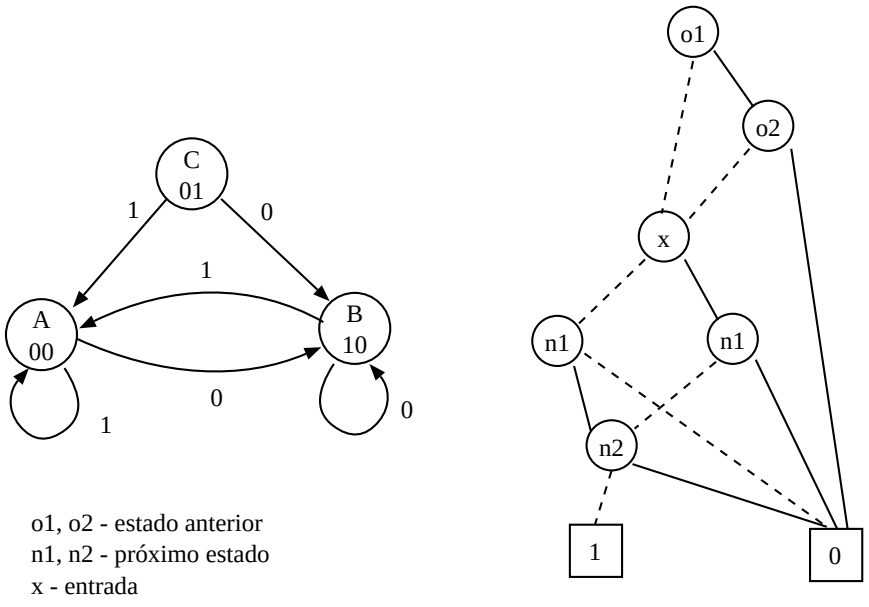



Figura 4: Diagrama de estados de uma máquina de estados e sua representação utilizando BDDs.

Em muitos casos não há interesse em representar a função $\eta$ a depender das entradas, pois pretende-se apenas saber se de um determinado estado tem um outro estado como seguinte. Ou seja, se existe uma entrada que pode fazer transitar a máquina do estado $\vec{s_i}$

para o estado $\vec{s_f}$. Assim, representa-se através de uma BDD a função $\eta'(\vec{s_i}, \vec{s_f})$, que terá o resultado 1 se $\vec{s_f}$ for sucessor e $\vec{s_i}$. Note-se que função $\eta'$ resulta da função característica $\eta$ após realizada uma operação de eliminação sobre as variáveis de entrada $\vec{x}$, isto e:

$$\eta'(s, s') = S_x[\eta(x, s, s')] = \exists x[\eta(x, s, s')]$$

Na figura 5 apresenta-se a BDD para a função $\eta'(\vec{s_i}, \vec{s_f})$ da máquina apresentada como exemplo.

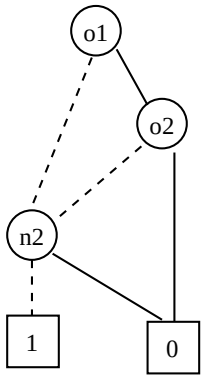



Figura 5: Representação da função $\eta'$.

Quando uma máquina de estados é pequena, como a apresentado no exemplo, a sua representação utilizando BDDs não é menor que a sua representação explicita. No entanto, para máquinas mais complexas e maiores a representação com BDDs pode ser consideravelmente menor.

## 3.3 Travessia de máquinas de estados

A travessia de uma máquina de estados $\mathcal{M}$ a partir de um conjunto de estados $C$ consiste em determinar os estados seguintes (travessia para a frente) ou estados anteriores (travessia para trás) desse conjunto $C$.

Seja $\vec{\delta} : B^a \rightarrow B^b$ a função que define o próximo estado de uma máquina de estados, sendo $a = ni + b$ ($ni$ é o número de entradas e $b$ o número de variáveis de estado). Seja $C_0 \subseteq B^b$ o conjunto dos estados inicias da máquina e $C_\infty \subseteq B^b$ o conjunto de estados atingíveis a partir de $C_0$. O conjunto $C_\infty$ pode ser calculado usando a seguinte formula iterativa:

$$C_{i+1} = C_i \cup \vec{\delta}(B^{ni} \times C_i)$$

A iteração termina quando $C_{i+1} = C_i$, sendo $C_\infty = C_i$. O conjunto $C_\infty$ contém todos os estados válidos da máquina, o conjunto de estados inválidos pode ser obtido complementando este conjunto.

A principal operação envolvida nestas iterações é o cálculo da imagem de um subconjunto. Existem dois métodos básicos para realizar a travessia da máquina de estados consoante a forma como é o calculada a imagem do subconjunto $C_i \times B^{ni}$ obtida por $\vec{\delta}$ (cálculo de $\vec{\delta}(C_i \times B^{ni})$ ).

Este dois métodos, que utilizam duas aproximações diferentes na representação da máquina de estados, serão apresentados nas secções seguintes.

### 3.3.1 Usando relações de transição

Este método de cálculo da imagem de um conjunto foi proposto em [Burch 90b].

**Definição 10** *Seja $\vec{\delta} : B^a \rightarrow B^b$ uma função booleana que define o próximo estado de máquina. A função caracteritica da* relação de transição *associada a $\vec{\delta}$, $\Delta : B^a \times B^b \rightarrow B$, é definida como $\Delta(x, y) = 1$ se e só se $(x, y) \in D = \{(x, y) \in B^a \times B^b : y = \vec{\delta}(x)\}$. O que é equivalente, em termos de operadores booleanos, a:*

$$\Delta(x, y) = \prod_{i=1}^{b} \left( y_i \equiv \vec{\delta}(x_i) \right)$$

A função $\Delta$, que é a função característica do conjunto $D$ ($\Delta = \chi_D$), pode ser usada para calcular a imagem de um subconjunto $A \subseteq B^a$. A imagem de $A$ obtida por $\vec{\delta}$ é definida como o conjunto $\vec{\delta}(A) = \{y : \exists x (x \in A) \wedge \Delta(x, y) = 1\}$. Se substituirmos o quantificador existencial pelo operador de eliminação, o $\wedge$ lógico pelo *and* booleano e representando os elementos de $A$ pela sua função característica $\chi_A$, então a função característica de $\vec{\delta}(A)$ ($\chi_{\vec{\delta}(A)}$) é representada em termos de operações booleanas por:

$$\chi_{\vec{\delta}(A)}(y) = S_x \left( \chi_A(x) \cdot \Delta(x, y) \right) \qquad \text{(ou, noutra notação: } \vec{\delta}(A) = S_x(A(x) \cdot \Delta(x, y))\text{)}$$

Esta fórmula pode também ser interpretada como a projecção em $B^b$ do conjunto que resulta da intercepção de $D$ com $A \times B^b$ ($D \cap (A \times B^b)$ ). As operações de eliminação e *and* booleano podem ser realizados numa única passagem na BDD que representam as funções.

A imagem inversa de um subconjunto $Z \subseteq B^b$ pode ser cálculada de forma idêntica:

$$\chi_{\vec{\delta}^{-1}(Z)}(x) = S_y \left( \chi_Z(y) \cdot \Delta(x, y) \right) \qquad \text{(ou, noutra notação: } \vec{\delta}^{-1}(Z) = S_y(Z(y) \cdot \Delta(x, y))\text{)}$$

Em [Touati 90] e [Malik 88] são apresentadas estratégias para aumentar a eficiência destes algoritmo. Enquanto que em [Touati 90] é apresentada uma forma de não calcular explicitamente $\Delta(x, y)$, em [Malik 88] são apresentadas heurísticas de ordenação das variáveis para reduzir o tamanho das BDDs utilizadas. Em [Burch 90a] é apresentada uma técnica conhecida com *iteative squaring* que permite reduzir o número de iterações de N (número de mínimo de iterações para determinar $C_\infty$) para $n = [\log_2 N]$.

### 3.3.2 Usando funções de transição

Este método de cálculo da imagem de um conjunto foi proposto por Coudert *et al.* [Coudert 89a, Coudert 90]. A sua principal vantagem reside no facto de não ser necessário construir a BDD para a relação de transição da máquina de estados.

**Definição 11** *Dada uma função booleana vectorial, $\vec{f} = [f_1, ..., f_b] : B^a \rightarrow B^b$, e um conjunto $C \subseteq B^a$ representado pela sua função característica, $c = \chi_C$, o* cofactor generalizado*, $f_c = [(f_1)_c, ..., (f_b)_c]$, é uma função de $B^a$ para $B^b$ cujo alcance é igual à imagem $C$ obtida por $\vec{f}$.*

Isto é, se a imagem de um subconjunto $C \subseteq B^a$ por $f$ for o conjunto $X$ ($f(C) = X$), então $f_c$ define-se como uma função em que a imagem de todo o domínio $B^a$ é igual a $X$ ($f_c(B^a) = f(C) = X$).

Para uma função de uma única saída, $f_i : B^a \rightarrow B$, o par $(f_i, c)$ pode ser interpretado como uma função não completamente especificada que define $f_i$ em $C$. Esta função tem como conjunto-ON $f_i \cdot c$ , como conjunto-OFF $\overline{f_i} \cdot c$ , e o conjunto para os quais o valor da função não interessa (*don't care*) é dado por $\overline{c}$. Assim, o alcance de $(f_i, c)$ coincide com a imagem de $c$ obtida por $f_i$. Com esta interpretação, o cofactor generalizado consiste numa selecção heurística de uma representação particular da função não completamente especificada $(f_i, c)$ cuja arepresentação utilizando BDDs seja a menor.

**Definição 12** *Seja $c : B^a \rightarrow B$ uma função booleana não nula com a seguinte ordenação das variáveis: $x_1 \prec x_2 \prec \cdots \prec x_a$. A* função de mapeamento *$\pi_c : B^a \rightarrow B^a$ é definida da seguinte forma:*

$$\pi_c(x) = \begin{cases} x, & se\ c(x) = 1 \\ x', & se\ c(x) = 0 \end{cases}$$

$$sendo\ x'\ um\ valor\ tal\ que: c(x') = 1 \wedge minimiza\ d(x, x')$$

$$com\ d(x, x') = \sum_{i=1}^{a} |x_i - x'_i| 2^{a-i}$$

A função de mapeamento assim definida mapeia todos os valores de $B^a$ nos elementos do conjunto $C$ (conjunto-ON de $c$), pelo que temos que $f_c = f \circ \pi_c$ (ou seja, $f_c(x) = f(\pi_c(x))$ ).

A vantagem de utilizarmos o cofactor generalizado é passarmos de um cálculo de uma imagem de um conjunto por uma função para o calculo do alcance de uma função. Na figura 6 mostra-se como a a imagem de um conjunto $C$ pode ser calculada como o alcance do cofactor dessa função:

$$f(C) = f(\pi_c(B^n)) = f_c(B^n)$$

O cofactor generalizado de uma função pode ser eficientemente calculado usando um única travessia de das BDDs que representam a função e $f$ e $c$. Um algoritmo para realizar esta travessia encontra-se em [Touati 90].

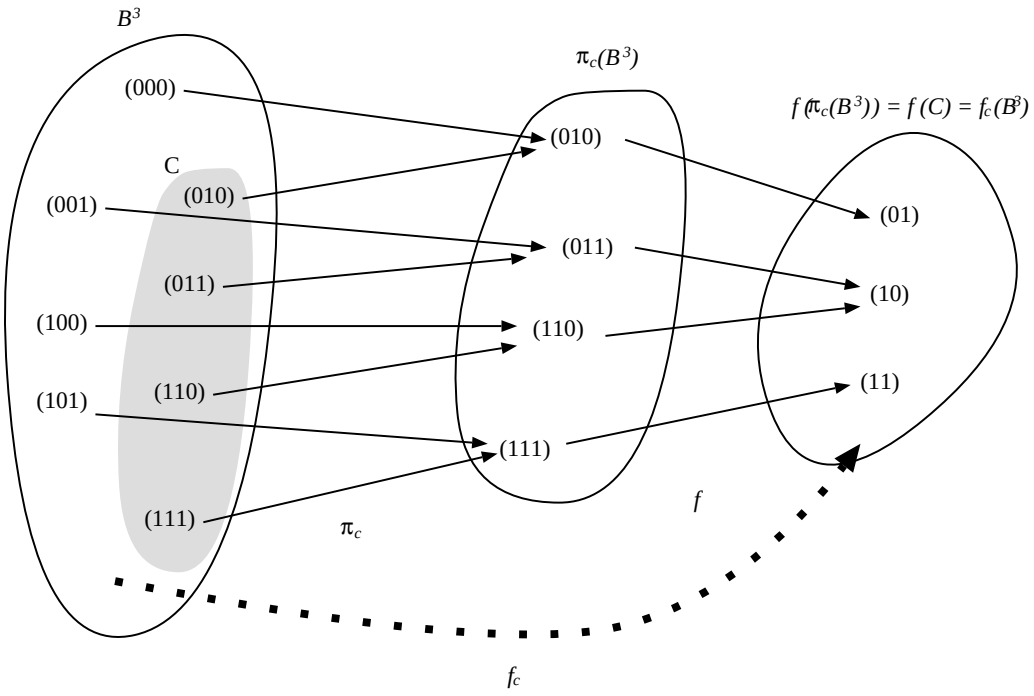



Figura 6: Utilização da função de mapeamento $\pi_c$ para calcular a imagem de um conjunto

O uso da função cofactor generalizado pode ser utilizado no cálculo iterativo que é feito para atravessar todos os estados da máquina e determinar $C_\infty$. Assim, cada passo de iteração envolve primeiro o cálculo do cofactor generalizado para obter $\vec{\delta}_{c_i} = [(\vec{\delta}_1)_{c_i}, (\vec{\delta}_2)_{c_i}, ..., (\vec{\delta}_b)_{c_i}]$, em que $c_i = \chi_{(B^{ni} \times C_i)}$, e depois o cálculo do alcance de $\vec{\delta}_{c_i}$.

Usando a decomposição de Shanon, a função característica do alcance de $\vec{\delta}_{c_i}$ (designada por $\mathcal{R}(\delta)$ para simpliccidade de notação) pode ser determinada utilizando a seguinte recursão:

$$\mathcal{R}(\delta)(y) = y_1 \cdot \mathcal{R}([\delta_2, ..., \delta_b]_{\delta_1}) + \overline{y_1} \cdot \mathcal{R}([\delta_2, ..., \delta_b]_{\overline{\delta_1}})$$

A eficiência deste método pode ser aumentada através do *caching* dos resultados intermédios e pelo cálculo em grupos separados, do alcance das funções $[\delta_1, \delta_2, ..., \delta_b]$ se a partir de qualquer passo da recursão estas tiverem conjunto de suporte disjuntos. A complexidade do algoritmo no pior dos casos reduz-se de $2^b$ para $2^{s_1} + \cdots + 2^{s_k}$, onde $(s_1, ..., s_k)$ são os tamanhos dos grupos $(s_1 + \cdots + s_k = b)$.

Este método pode ser usado para determinar a imagem de um conjunto, mas não para determinar uma imagem inversa, como acontece com o método que usa relação de transição da máquina de estados.

Em ambos os métodos de cálculo, usando relações de transição ou através da determinação de alcances, quando se pretende calcular o estados atingíveis na iteração $i$, $C_{i+1}$, não é necessário calcular toda a imagem de $C_i$. Basta apenas considerar o subconjunto de estados, $S_i$, que foram acrescentados na iteração anterior, $C_i - C_{i-1} \subseteq S_i \subseteq C_i$. Como só os estados fora de $C_{i-1}$ vão ser importantes, a função não completamente especificada $(C_i, \overline{C_{i-1}})$ pode ser usada para determinar quais os novos estados que vão ser atingíveis. Assim, é possível deminuir o tamanho da BDD que representa o conjunto dos estados actuais considerados para cada iteração, aumentado a eficiência dos algoritmos.

Em [Touati 90] são comparados os diferentes algoritmos para travessia de máquinas de estados (utilizando os circuitos sequenciais de *benchmarking* das ISCAS). Na maioria dos casos o método da relação de transição moustrou-se mais eficiente que o método das funções de transição.

# 4 Verificação de fórmulas de lógica temporal

## 4.1 Lógica Temporal - Terminologia

A lógica temporal foi desenvolvida para raciocinar sobre a ordenação de acontecimentos no tempo sem no entanto introduzir explicitamente a variável tempo.

O caso mais geral de lógica temporal é denominada de CTL* (Computed Tree Logic) [Clarke 87]. Esta lógica é constituída por fórmulas de estado, que são válidas num estado específico, e por fórmulas de caminho, que são válidas ao longo de uma sequência de estados (caminho).

**Definição 13** *As* fórmulas de estado *são constituídas por:*

- *uma proposição atómica,*
- $\overline{f}$, $f \wedge g$ *ou* $f \vee g$, *com* $f$ *e* $g$ *fórmulas de estado,*
- $A(f)$ *ou* $E(f)$, *sendo* $f$ *uma fórmula de caminho*

$A(f)$ é verdadeira num estado, quando $f$ é verdadeira ao longo de todos os caminhos que partem desse estado. $E(f)$ é verdadeira num estado, quando existe algum caminho, com início nesse estado, ao longo do qual $f$ é verdadeira.

**Definição 14** *As* fórmulas de caminho *são constituídas por:*

- *uma fórmula de estado,*
- $\overline{f}$, $f \wedge g$, $f \vee g$, $Gf$, $Ff$, $Xf$, *ou* $fUg$,*com* $f$ *e* $g$ *fórmulas de caminho*

Uma fórmula de caminho, constituída por uma fórmula de estado, é verdadeira se a fórmula de estado for verdadeira no primeiro estado. A fórmula $Gf$ significa "sempre $f$" e é verdadeira se $f$ for verdadeira em todos os estados do caminho. A fórmula $Ff$, que se lê "eventualmente $f$", é verdadeira se existir ao longo do caminho um estado em que

$f$ seja verdadeira. $Xf$ significa o "próximo estado $f$" e é verdadeira se $f$ for verdadeira no segundo estado do caminho. A fórmula $fUg$, que significa "$f$ até que $g$", é verdadeira se existir um estado no caminho em que $g$ é verdadeira e em todos os estados anteriores $f$ foi verdadeira.

A lógica mais usada na verificação de circuitos digitais (máquinas de estados ) é a lógica CTL. Esta lógica é um subconjunto da lógica CTL* em que cada quantificador de caminho ($A$ ou $E$) é seguido por um dos operadores temporais: $G$, $F$, $X$ ou $U$.

## 4.2 Algoritmos de verificação

A tarefa de verificar uma fórmula sobre um grafo de transições de estados (que pode ser a representação de uma máquina de estados) é designado genericamente por verificação de modelos (*model checking*).

Usando técnicas de travessia de grafos, a complexidade dos algoritmos de verificação é linear com o tamanho do grafo e da fórmula. No entanto, o tamanho da representação do grafo de transições pode explodir se este for extraído de um circuito com elevado número de elementos de memória. Assim, é necessário usar algoritmos que utilizem BDDs para representar o grafo de transições evitando a sua representação/enumeração explicita. Este tipo de verificação chama-se de verificação simbólica de modelos (*symbolic model checking*).

A verificação simbólica de modelos utilizando fórmulas de CTL é feita determinando o conjunto de estados no qual a fórmula CTL é verdadeira, verificando-se depois se este conjunto é equivalente ao conjunto de todos os estados atingíveis.

Para determinar o conjunto de estados em que uma fórmula é verdadeira começa-se por aplicar um conjunto de reduções [Clarke 87, Aelten 92, Hojati 93]. Os quantificador de caminho universal, $A$, pode ser reduzido ao quantificador existencial $E$:

$$Af = \overline{E\overline{f}}$$

$$AXf = \overline{EX\overline{f}}$$

$$AFf = \overline{EG\overline{f}}$$

$$AGf = \overline{EF\overline{f}}$$

$$A(fUg) = \overline{E[\overline{g}U(\overline{f}\cdot\overline{g})]} \wedge \overline{EG\overline{g}}$$

Fórmulas com os operadores temporais $F$ e $G$ podem ser substituídas por fórmulas utilizando apenas o operador $U$:

$$Ff = 1Uf$$

$$Gf = \overline{F\overline{f}} = \overline{1U\overline{f}}$$

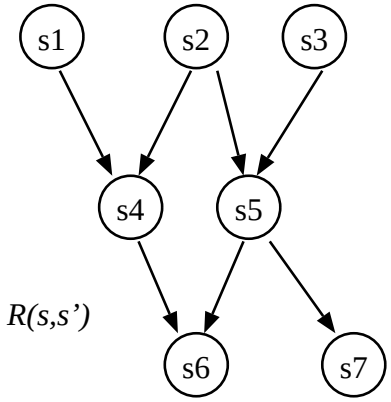
Assim, é possível determinar o valor de qualquer fórmula CTL se se tiver algoritmos para determinar o valor de $EXf$ e $E(fUg)$.

O algoritmo para o cálculo de fórmulas é definido através da função $Bdd$ que com argumento uma fórmula $f$. Esta função, $Bdd(f)$, retorna uma BDD que é verdadeira num dado estado se a fórmula $f$ é verdadeira nesse estado. A função $f$ se for uma proposição atómica $p$ é representada BDD de $p$. A relação de transição da máquina de estados, $R(s, s')$, é também representada por uma BDD, o estado $s'$ é o sucessor de $s$ se a relação for satisfeita.

A fórmula $EXf$ é verdadeira num dado estado se e só se existir um estado sucessor em que $f$ seja verdadeira. Ou seja, se o estado actual for $s$ então existe um estado $s'$ que satisfaz $R(s, s')$ e $Bdd(f)$ é verdadeira. Usando o operador de quantificação booleana podemos exprimir esta condição como:

$$Bdd(EXf)(s) = \exists s'[Bdd(f)(s') \wedge R(s, s')]$$

Note-se que $Bdd(EXf)$ é obtido através do cálculo da imagem inversa do conjunto de todos os estados em que a $Bdd(f)$, e portanto $f$, é verdadeira. Se se tivesse a usar a fórmulas CTL no passado, em vez de no futuro, $EX^{-}f$ significaria todos os estados em que têm um antecessor no qual $f$ é verdadeira. Nesta caso, $BDD(EXf)$ resultaria do cálculo da imagem do conjunto de todos os estados no qual $f$ fosse verdadeira. A figura 7 exemplifica o significado da fórmula $EXf$ e a diferença entre a utilização de fórmulas CTL no passado e no futuro.




$$EX(S = s5) = \delta^{-1}(\{s5\}) = \{s2, s3\}$$

$$EX^{-}(S = s5) = \delta(\{s5\}) = \{s6, s7\}$$

Figura 7: Significado da fórmula $EXf$.

A formula $E[fUg]$ significa que existe um caminho a partir do estado actual no qual: $g$ é verdadeiro algures no futuro e $f$ é verdadeiro em todos os estados que lhe antecedem. Isto significa que $g$ é verdadeiro no estado actual, ou então, $f$ é verdadeiro e existe um estado sucessor no qual $E[fUg]$ e verdadeiro. $E[fUg]$ pode ser calculado usando o seguinte algoritmo:

1. Determinar o conjunto de estados no qual $g$ é verdadeira.
2. Adicionar a esses estados o conjunto de estados que lhes deram origem e no qual $f$ é verdadeira.

3. Se novos estados tiverem sido adicionados repetir o passo 2, caso contrario o conjunto de estados obtido satisfaz a fórmula $E[fUg]$

Este procedimento pode ser formalmente definido da seguinte forma: $E[fUg]$ pode ser calculado como o ponto-fixo-mínimo da seguinte equação recursiva:

$$\mathbf{E}[\mathbf{fUg}] = g \vee [f \wedge EX(\mathbf{E}[\mathbf{fUg}])]$$

Por ponto-fixo-mínimo entende-se o conjunto mínimo que satisfaz a equação recursiva anterior. Para obter este conjunto é necessário efectuar repetidos cálculos de imagens inversas de conjuntos até se atingir a convergência para o conjunto que representa os estados em que $E[fUg]$ é verdadeira.

# Referências


[Aelten 92] Filip Van Aelten. *Automatic Procedure for the Behavioral Verification of Digital Designs.* PhD thesis, Massachusetts Institute of Technology, May 1992.

[Akers 78] S. B. Akers. Binary Decision Diagrams. C(27):209–516, June 1978.

[Bose 89] S. Bose and A. L. Fisher. Automatic Verification of Synchronous Circuits Using Symboloic Logic Simulation and Temporal Logic. In *International Workshop on Applied Formal Methods for Correct VLSI Design*, volume VLSI Design Methods-II, pages 151–158. IFIP WG10.2/WG10.5, 1989.

[Brace 90] Karl S. Brace, Richaed L. Rudell, and Randal E. Bryant. Efficent Implementation of a BDD Package. pages 40–45, June 1990.

[Bryant 86] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. pages 677–691, June 1986.

[Bryant 95] Randal E. Bryant. Binary Decision Diagrams and Beyond: Enabling Technologies for Formal Verification. 1995.

[Burch 90a] J. R. Burch, E. M. Clarke, K. L. MacMillan, D. L. Dill, and J. Hwang. $10^{20}$ States and Beyond. In *LICS'90:* $5^{th}$ *Annual IEEE Symposium on Logic in Computer Science*, pages 428–439. IEEE, Novenmber 1990.

[Burch 90b] J. R. Burch, E. M. Clarke, K. L. McMillan, and David L. Dill. Sequential Circuit Verification Using Symbolic Model Checking. pages 46–51, June 1990.

[Burch 94] Jerry R. Burch, Edmund M. Clarke, David E. Long, Kenneth L. McMillan, and David L. Dill. Symbolic Model Checking for Sequential Circuit Verification. 13(4):401–424, April 1994.

[Clarke 87] E. M. Clarke and O. Grumberg. Research on Automatic Verification of Finite-State Concurrent Systems. In *Annual Review of Computer Science*, volume 2, pages 269–290. Annual Reviews Inc., 1987.

[Coudert 89a] O. Coudert, C. Berthet, and J. C. Madre. Verification of Syncronous Sequencial Machines Using Symbolic Execution. In *Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, June 1989.

[Coudert 89b] O. Coudert, C. Berthrt, and J. C. Madre. Verification of Sequential Machines Using Boolean Functional Vectors. In *IFIP - International Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128. IMEC, June 1989.

[Coudert 90] O. Coudert, J. C. Madre, and C. Berthet. Verifying Temporal Properties of Sequencial Machines Without Building their State Diagrams. *Workshop on Computer Aided Verification*, June 1990.

[Coudert 91] O. Coudert and J. C Madre. Symbolic computation of the valid states of a sequential machine: algorithms and discussion. In *International Workshop on Formal Methods for Correct VLSI Design*. ACM/IFIP WG10.2, January 1991.

[Hojati 93] R. Hojati, T. R. Shiple, R. K. Brayton, and R. P. Kurshan. A Unified Approach to Language Containment and Fair CTL Model Checking. pages 475–481, June 1993.

[Lee 59] C. E. Lee. Representation of Switching Circuits by Binary Decision Programs. *Bell System Technical Journal*, 38:985–999, July 1959.

[Malik 88] S. Malik et al. Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment. pages 6–9, 1988.

[Touati 90] H. J. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit State Enumeration of Finite State Machines using BDD's. pages 130–133, 1990.