Binary Decision Diagrams (Einführung)

Binary Decision Diagrams (BDDs) sind bestimmte Graphen, die als Datenstruktur für die kompakte Darstellung von booleschen Funktionen benutzt werden.

BDDs wurden von R. Bryant 1986 eingeführt.

BDDs werden sehr häufig benutzt, um Äquivalenzprobleme zwischen aussagenlogischen Formeln zu lösen.

Sehr wichtig im Bereich Hardwareentwurf und Hardwareoptimierung.

Graphen

Ein (endlicher) gerichteter Graph ist ein Paar $G = (V, E)$, wobei V eine (endliche) Menge von Knoten und $E \subseteq V \times V$ eine Menge von Kanten ist.

Ein Knoten wird graphisch durch einen Punkt dargestellt.
Eine Kante (v, v') wird graphisch durch einen Pfeil von v nach v' dargestellt.

Die Vorgänger eines Knotens v sind die Knoten v', für die es Kanten (v', v) gibt. Analog werden die Nachfolger definiert.

Pfade und Zyklen

Ein Pfad von v nach v' ist eine nichtleere Sequenz

$$(v, v_1)(v_1, v_2)(v_2, v_3) \ldots (v_{n-1}, v_n)(v_n, v')$$

von Kanten.

Ein Knoten v' ist aus v erreichbar, wenn $v = v'$ oder es einen Pfad von v nach v' gibt.

Ein Zyklus ist ein Pfad von einem nach demselben Knoten.

Azyklische Graphen und ihre Untergraphen

Ein azyklischer Graph ist ein Graph ohne Zyklen.

Sei $G = (V, E)$ ein azyklischer Graph und sei $v \in V$ ein Knoten. Der Graph $G_v = (V', E')$ wird wie folgt definiert:

- V' enthält alle Knoten von V, die aus v erreichbar sind.
- E' enthält alle Kanten (v_1, v_2) mit $v_1 \in V'$.

Wir nennen G_v einen Untergraph von G. Wenn $V = \{v_1, \ldots, v_n\}$, dann ist $\{G_{v_1}, \ldots, G_{v_n}\}$ die Menge der Untergraphen von G.

Bäume und Wälder

Ein Baum ist ein gerichteter Graph, der die folgenden Eigenschaften erfüllt:

(1) Der Graph enthält keine Zyklen.

(2) Alle Knoten haben höchstens einen Vorgänger

(3) Es gibt genau einen Knoten ohne Vorgänger.

Der Knoten ohne Vorgänger heißt die Wurzel des Graphen. Die Knoten ohne Nachfolger heißen Blätter. Die Nachfolger eines Knotens sind seine Kinder.
Ein Wald ist ein Graph, der (1) und (2) erfüllt, aber nicht unbedingt (3), d.h. ein Wald darf mehrere Wurzeln haben.

Boolesche Funktionen

Eine boolesche Funktion der Arität $n \geq 1$ ist eine Funktion $\{0,1\}^n \to \{0,1\}$.

Beispiele:

$$\text{oder}(x_1, x_2) = \begin{cases} 1 & \text{wenn } x_1 = 1 \text{ oder } x_2 = 1 \\ 0 & \text{wenn } x_1 = 0 \text{ und } x_2 = 0 \end{cases}$$

$$\text{if\_then\_else}(x_1, x_2, x_3) = \begin{cases} x_2 & \text{wenn } x_1 = 1 \\ x_3 & \text{wenn } x_1 = 0 \end{cases}$$

Z.B.: $\text{if\_then\_else}(1, 0, 1) = 0$, $\text{if\_then\_else}(0, 0, 1) = 1$

$$\text{summe}(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{wenn } x_1 + x_2 = x_3 x_4 \\ 0 & \text{sonst} \end{cases}$$

Z.B.: $\text{summe}(1, 1, 1, 0) = 1$ (denn $1 + 1 = 10$),
$\text{summe}(0, 0, 0, 1) = 0$ (denn $0 + 0 = 00$).

$$\text{mehrheit}_n(x_1, \ldots, x_n) = \begin{cases} 1 & \text{wenn die Mehrheit der Eingaben} \\ & x_1, \ldots, x_n \text{ den Wert } 1 \text{ hat} \\ 0 & \text{sonst} \end{cases}$$

Z.B.: $\text{mehrheit}_4(1, 1, 0, 0) = 0$, $\text{mehrheit}_3(1, 0, 1) = 1$

$$\text{parität}_n(x_1, \ldots, x_n) = \begin{cases} 1 & \text{wenn die Anzahl der Eingaben } x_1, \ldots, x_n \\ & \text{mit dem Wert } 1 \text{ gerade ist} \\ 0 & \text{sonst} \end{cases}$$

Z.B.: $\text{parität}_3(1, 0, 1) = 1$, $\text{parität}_2(1, 0) = 0$

Formeln und boolesche Funktionen

Sei F eine Formel, und sei n eine Zahl mit der Eigenschaft, dass alle atomaren Formeln, die in F vorkommen, zur Menge $\{A_1, \ldots, A_n\}$ gehören.

Beispiel: $F = A_1 \wedge A_2$, $n = 2$, aber auch $n = 3$!

Wir definieren die boolesche Funktion $f_F^n \colon \{0,1\}^n \to \{0,1\}$:

$f_F^n(x_1, \ldots, x_n) =$ Wahrheitswert von F unter der Belegung, die den Variablen $A_1, \ldots, A_n$ die Werte $x_1, \ldots, x_n$ zuordnet.

Beispiel: Für $F = A_1 \wedge A_2$:

$$\begin{aligned} f_F^2(0,1) &= \text{Wert von } 0 \wedge 1 = 0 \\ f_F^3(0,1,1) &= \text{Wert von } 0 \wedge 1 = 0 \end{aligned}$$

Bemerkung: Wenn $\{A_1, \ldots, A_n\}$ die atomaren Formeln sind, die in F vorkommen, dann ist f_F^n im Wesentlichen die übliche Wahrheitstabelle von F.

Konvention: Wir schreiben z.B. $f(x_1, x_2, x_3) = x_1 \vee (x_2 \wedge \neg x_1)$. Damit meinen wir, dass $f = f_F^3$ für die Formel $F = A_1 \vee (A_2 \wedge \neg A_1)$.

Fakt: Seien F_1 und F_2 zwei Formeln und sei n eine Zahl mit der Eigenschaft, daß alle atomare Formeln, die in F_1 oder F_2 vorkommen, zur Menge $\{A_1, \ldots, A_n\}$ gehören. Dann gilt $f^n_{F_1} = f^n_{F_2}$ gdw. $F_1 \equiv F_2$.

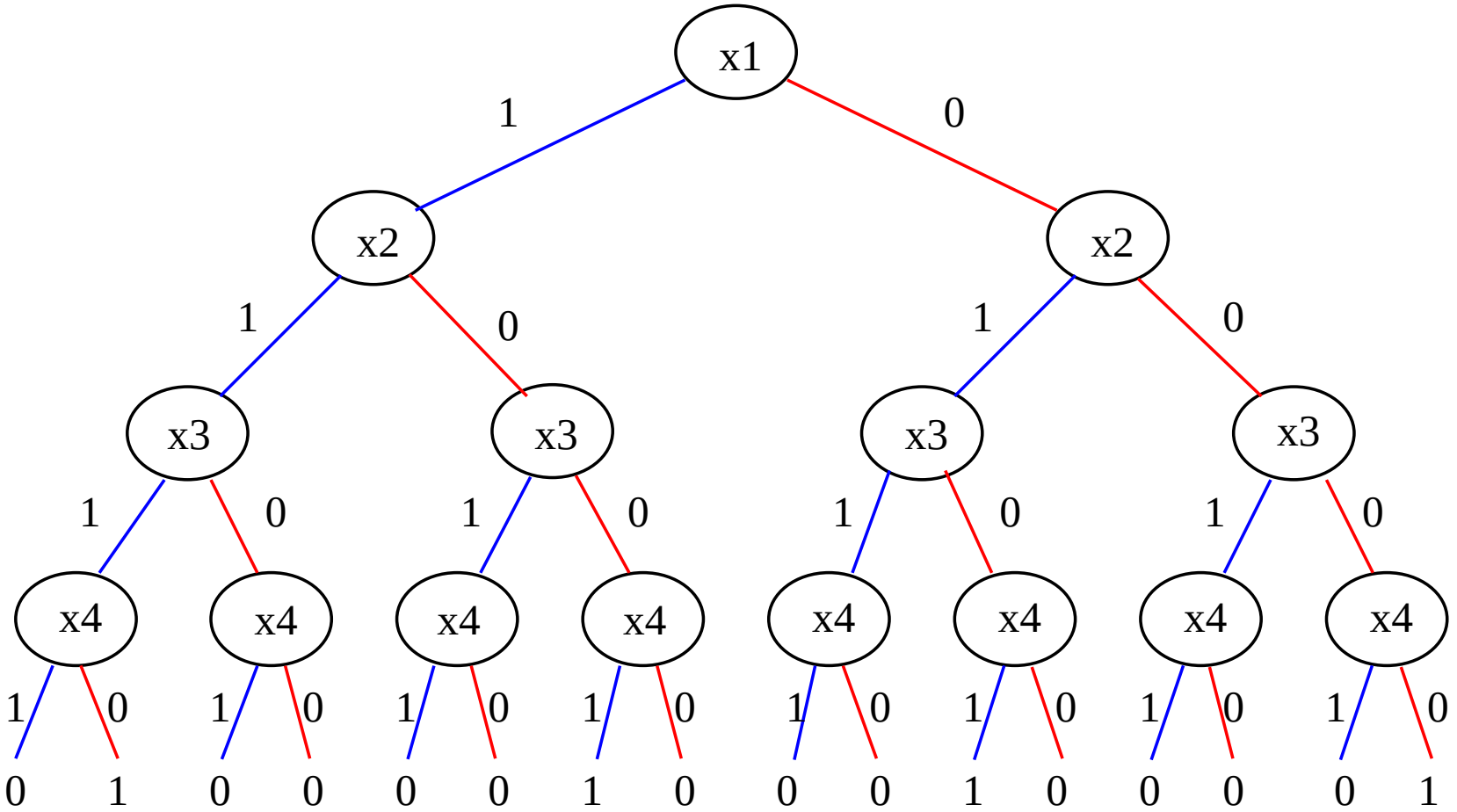
Beispiel: $F_1 = A_1$, $F_2 = A_1 \wedge (A_2 \vee \neg A_2)$.

$$\begin{array}{lcl} f^2_{F_1}(0,0) & = & 0 \quad = \quad f^2_{F_2}(0,0) \\ f^2_{F_1}(0,1) & = & 0 \quad = \quad f^2_{F_2}(0,1) \\ f^2_{F_1}(1,0) & = & 1 \quad = \quad f^2_{F_2}(1,0) \\ f^2_{F_1}(1,1) & = & 1 \quad = \quad f^2_{F_2}(1,1) \end{array}$$

Konvention: Die Konstanten $\mathbf{0}$ und $\mathbf{1}$ werden als (die einzigen) boolesche Funktionen der Arität 0 aufgegasst.

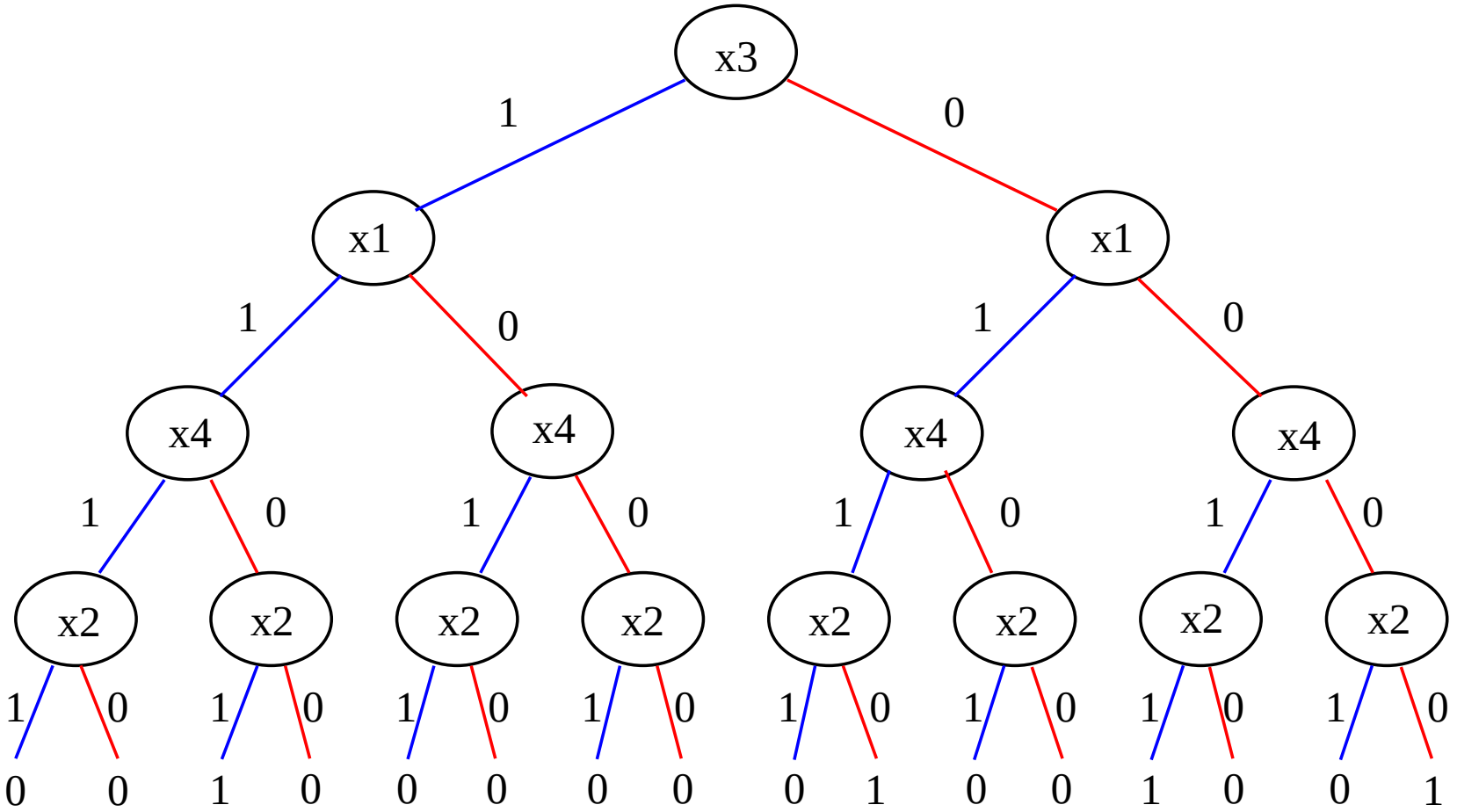
summe als binärer Entscheidungsbaum

Eine boolesche Funktion kann als binärer Entscheidungsbaum dargestellt werden



Variablenordnungen

Ein Entscheidungsbaum kann die Variablen in einer anderen Reihenfolge betrachten als in der Funktion vorgegeben.



Eine Variablenordnung ist eine Bijektion

$$b\colon \{1, \ldots, n\} \to \{x_1, \ldots, x_n\}$$

Wir sagen, dass $b(1), b(2), b(3), \ldots, b(n)$ die erste, zweite, dritte, ..., n-te Variable entsprechend der Ordnung b ist.

Für die Bijektion $b(1) = x_{i_1}, \ldots b(n) = x_{i_n}$ benutzen wir die Notation

$$x_{i_1} < x_{i_2} < \ldots < x_{i_n} \, .$$

Binäre Entscheidungsbäume

Ein binärer Entscheidungsbaum für die Variablenordnung $x_{i_1} < \ldots < x_{i_n}$ ist ein Baum, der die folgenden Eigenschaften erfüllt:

(1) Alle Blätter sind mit 0 oder 1 beschriftet.

(2) Alle anderen Knoten sind mit einer Variablen beschriftet und haben genau zwei Kinder, das 0-Kind und das 1-Kind. Die Kanten, die zu den Kindern führen, sind mit 0 bzw. mit 1 beschriftet.

(3) Wenn die Wurzel kein Blatt ist, dann ist sie mit x_{i_1} beschriftet.

(4) Wenn ein Knoten mit x_{i_n} beschriftet ist, dann sind seine beide Kinder Blätter.

(5) Wenn ein Knoten mit x_{i_j} beschriftet ist und $j < n$, dann sind seine beide Kinder mit $x_{i_{j+1}}$ beschriftet.

Jeder Pfad eines binären Entschiedungsbaums entspricht einer Belegung der Variablen $x_{i_1}, \ldots x_{i_n}$ und umgekehrt.

Die von einem binären Entscheidungsbaum T dargestellte boolesche Funktion f_T wird folgendermaßen definiert:

$f_T(x_1, \ldots, x_n) =$ Beschriftung des Blatts, das durch den Pfad, der der Belegung $x_{i_1}\, x_{i_2}\, \ldots\, x_{i_n}$ entspricht, erreicht wird.

Ein binärer Entscheidungswald ist ein Wald von binären Entscheidungsbäumen mit derselben Variablenordnung. Ein Entscheidungswald stellt die Menge der Funktionen dar, die von den Wurzeln der Bäume dargestellt werden.

Binary Decision Diagrams (informell)

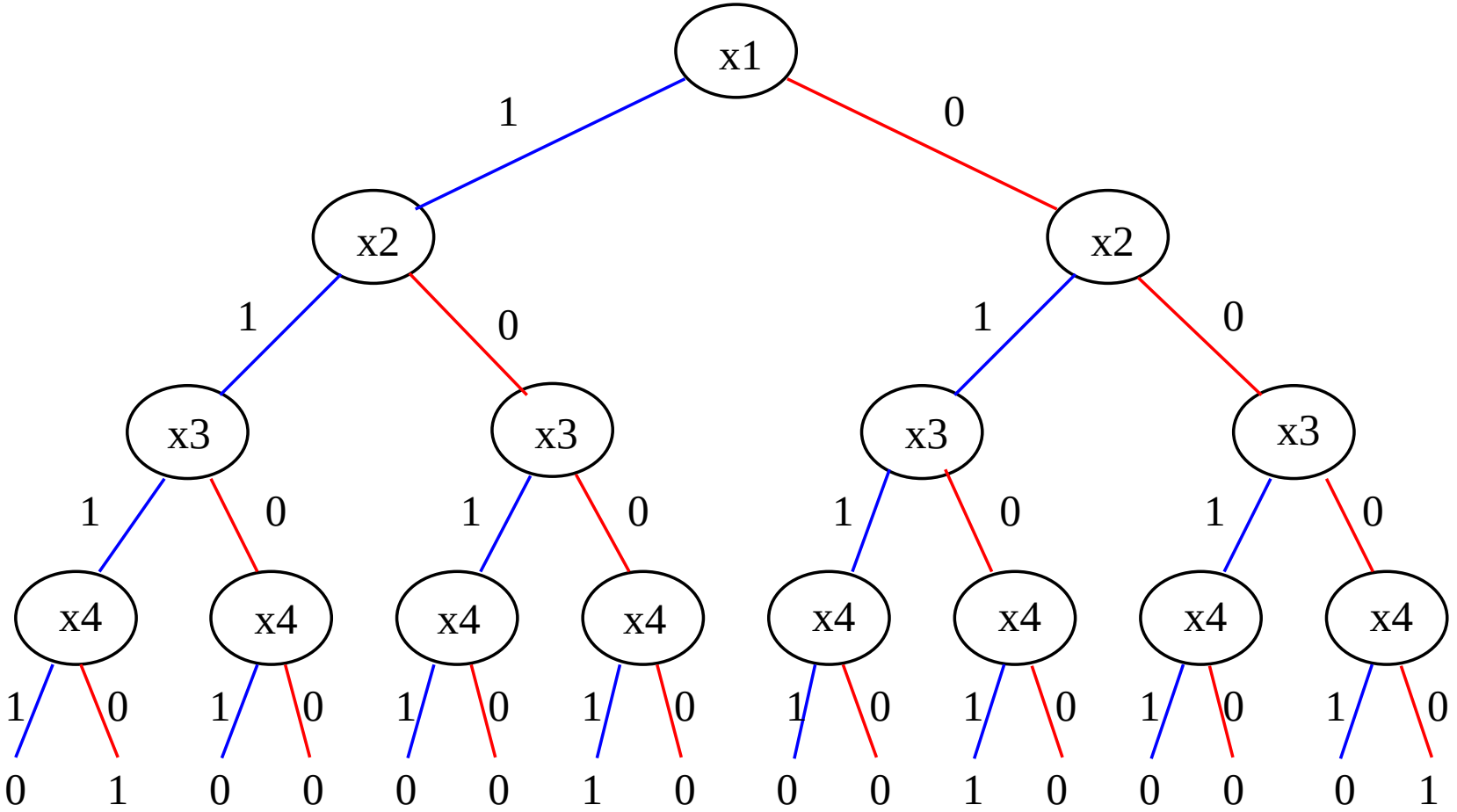
Ein BDD (multiBDD) ist eine "komprimierte Darstellung" eines binären Entscheidungsbaums (Entscheidungswalds).

Ein BDD (multiBDD) kann aus einem Entscheidungsbaum (-wald) durch wiederholte Anwendung zweier Komprimierungsregeln gewonnen werden (siehe Beispiel auf den nächsten Folien):

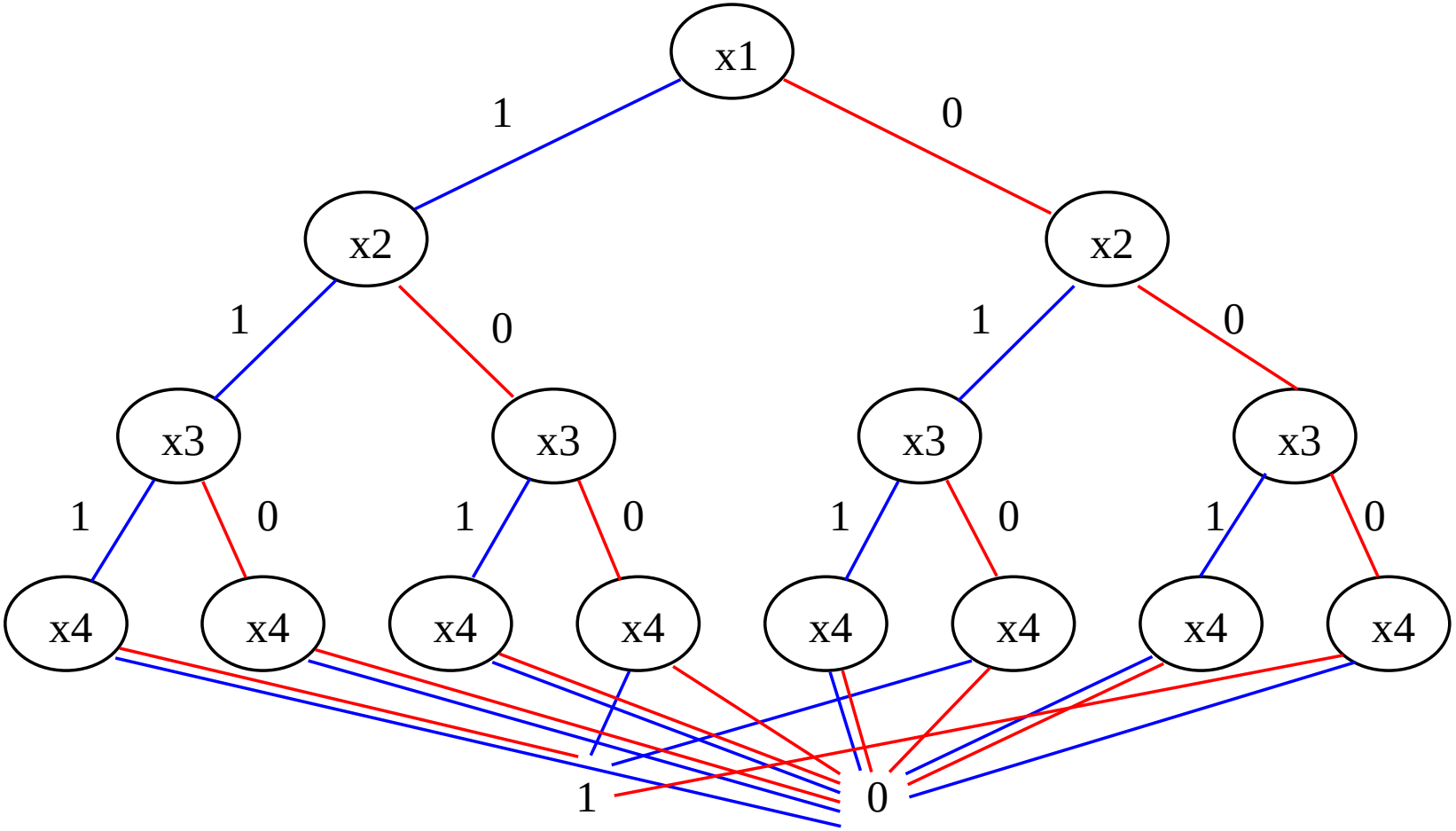
- Regel 1: Gleiche Unterdiagramme wiederverwenden.
- Regel 2: Innere Knoten entfernen, bei denen das 0-Kind und das 1-Kind derselbe Knoten ist (redundante Knoten).

Diese Regeln werden angewendet, bis alle Unterdiagramme verschieden sind und es keine redundanten Knoten gibt.

Beispiel: Unterdiagramme wiederverwenden

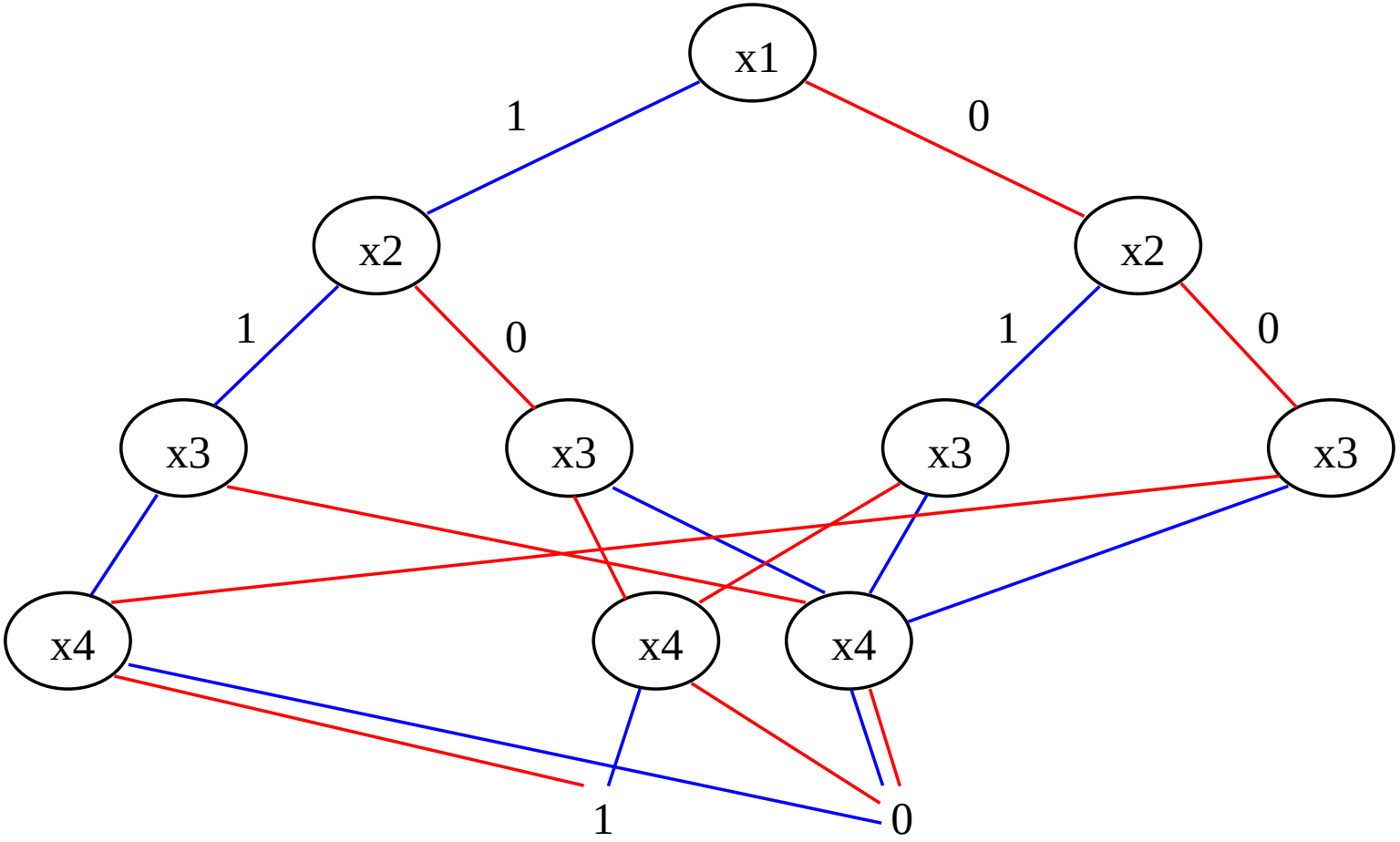


Beispiel: Unterdiagramme wiederverwenden



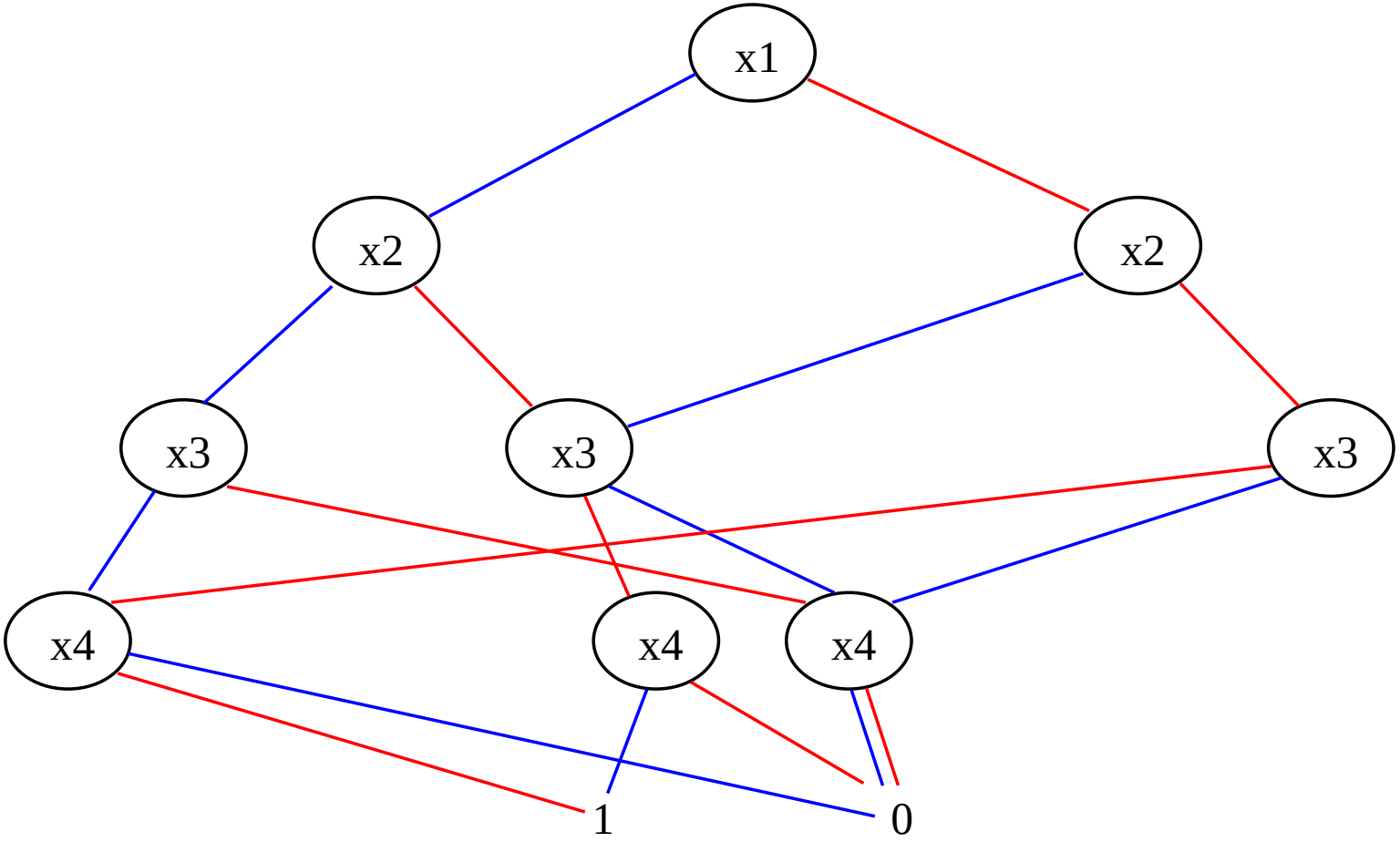
Alle 0- und 1-Knoten zusammengefasst.

Beispiel: Unterdiagramme wiederverwenden



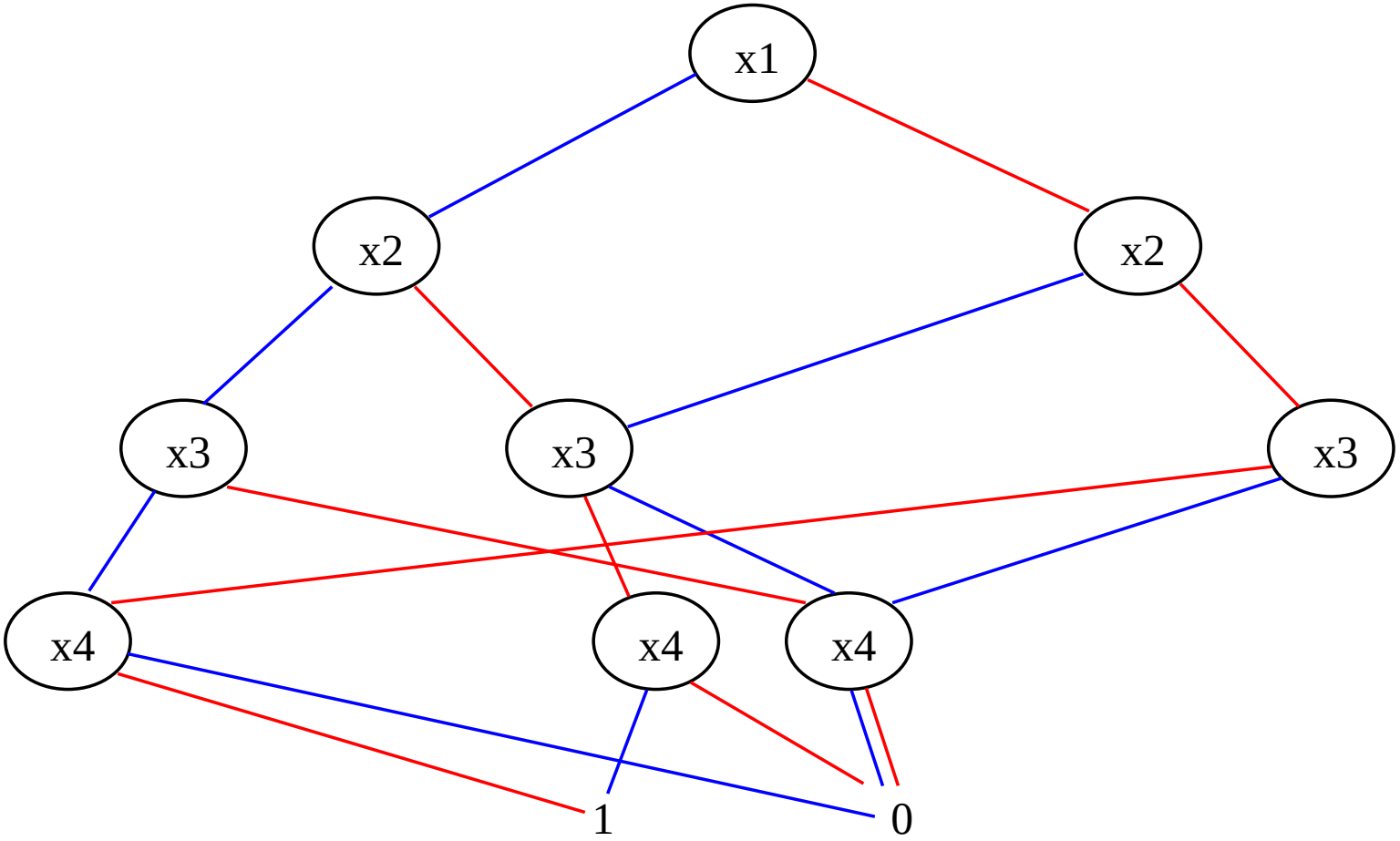
Gleiche x_4-Knoten zusammengefasst.

Beispiel: Unterdiagramme wiederverwenden

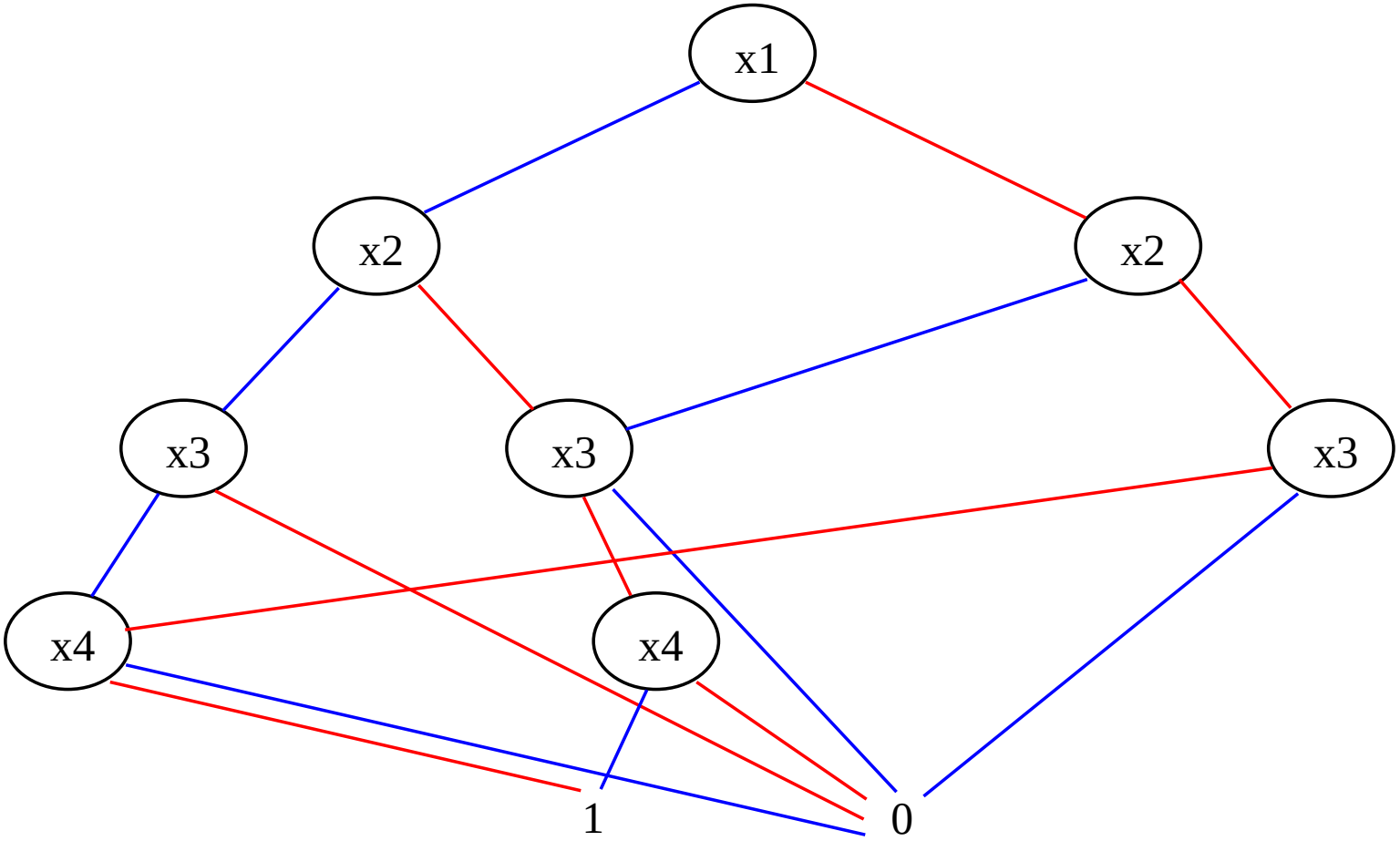


Gleiche x_3-Knoten zusammengefaßt; fertig.

Beispiel: Redundante Knoten entfernen



Beispiel: Redundante Knoten entfernen



x_4-Knoten entfernt

Formale Definition von BDDs

Ein BDD für eine gegebene Variablenordnung ist ein azyklischer Graph, der die folgenden Eigenschaften erfüllt:

(1) Es gibt genau einen Knoten ohne Vorgänger (die Wurzel)

(2) Es gibt einen oder zwei Knoten ohne Nachfolger. Sie sind mit 0 oder 1 beschriftet.

(3) Alle anderen Knoten sind mit einer Variable beschriftet und haben (genau) zwei verschiedene Kinder, das 0-Kind und das 1-Kind. Die Kante, die zu dem 0-Kind bzw. dem 1-Kind führt, ist mit 0 bzw. mit 1 beschriftet.

(4) Ein Kind eines Knotens ist mit 0, 1, oder mit einer größeren Variablen als die Variablen seiner Vorgänger beschriftet.

(5) Alle Untergraphen des Graphen sind verschieden.

MultiBDDs

Ein multiBDD ist ein azyklischer Graph, der (2)-(5) erfüllt und einige ausgezeichnete Knoten enthält, die wir die Wurzeln nennen.

Jeder Knoten ohne Vorgänger ist eine Wurzel. Knoten mit Vorgängern dürfen jedoch auch Wurzeln sein.

Ein multiBDD stell eine Menge von booleschen Funktionen dar, eine für jede Wurzel.

Bemerkungen

Bemerkung: Ein Untergraph eines BDDs ist wieder ein BDD.
Bemerkung: Die Funktion $\text{true}_n(x_1, \ldots, x_n)$ mit

$$\text{true}_n(x_1, \ldots, x_n) = 1 \text{ für alle } x_1, x_n \in \{0,1\}^n$$

wird für jedes $n \geq 1$ und für jede Variablenordnung durch das BDD dargestellt, das aus einem einzigen, mit 0 beschrifteten Knoten besteht.

Analoges gilt für die Funktion $\text{false}_n(x_1, \ldots, x_n)$

Einfluß der Variablenordnung

Die Wahl der Variablenordnung kann die BDD-Größe erheblich beeinflussen.

Beispiel:

$$f(x_1, \ldots, x_n, x_{n+1}, \ldots, x_{2n}) = (x_1 \leftrightarrow x_{n+1}) \wedge (x_2 \leftrightarrow x_{n+2}) \wedge \cdots \wedge (x_n \leftrightarrow x_{2n})$$

Größe wächst exponentiell in n für

$x_1 < \cdots < x_n < x_{n+1} < \cdots < x_{2n}$.

Größe wächst linear in n für

$x_1 < x_{n+1} < x_2 < x_{n+2} < \ldots < x_n < x_{2n}$.

Problem in der Praxis: Wahl einer günstigen Variablenordnung.

Eindeutigkeit von BDDs

Wir zeigen, dass es für jede boolesche Funktion und für jede Variablenordnung ein einziges BDD gibt, dass die Funktion darstellt.

Dafür zeigen wir allgemeiner (aber einfacher!), dass es für jede Menge boolescher Funktionen derselben Arität und für jede Variablenordnung ein einziges multiBDD gibt, dass die Menge darstellt.

Die Funktionen $f[0]$ und $f[1]$

Lemma I: Sei f eine boolesche Funktion der Arität $n \geq 1$. Es gibt genau zwei boolesche Funktionen $f[0]$ und $f[1]$ der Arität $(n-1)$ mit

$$f(x_1, \ldots, x_n) = (\neg x_1 \wedge f[0](x_2, \ldots, x_n)) \vee (x_1 \wedge f[1](x_2, \ldots, x_n)) \qquad (1)$$

Beweis: Die Funktionen $f[0]$ und $f[1]$ definiert durch
$f[0](x_2, \ldots, x_n) = f(0, x_2, \ldots, x_n)$ und
$f[1](x_2, \ldots, x_n) = f(1, x_2, \ldots, x_n)$ erfüllen die Gleichung (1).
Seien f_0 und f_1 beliebige Funktionen die (1) erfüllen. D.h.
$f(x_1, \ldots, x_n) = (\neg x_1 \wedge f_0(x_2, \ldots, x_n)) \vee (x_1 \wedge f_1(x_2, \ldots, x_n))$.
Aus den Eigenschaften von $\vee$ und $\wedge$ folgt
$f(0, x_2, \ldots, x_n) = f_0(x_2, \ldots, x_n)$ und zusammen mit
$f(0, x_2, \ldots, x_n) = f[0](x_2, \ldots, x_n)$
gilt $f_0 = f[0]$. Analog zeigt man $f_1 = f[1]$.

Sei $f\colon \{0,1\}^n \to \{0,1\}$ eine boolesche Funktion, sei B ein BDD mit der Variablenordnung $x_1 < x_2 < \ldots < x_n$, und sei v die Wurzel von B. Definiere die Knoten $v[0]$ und $v[1]$ wie folgt:

(1) Wenn v mit x_1 beschriftet ist, dann sind $v[0]$ und $v[1]$ das 0-Kind und das 1-Kind von v.

(2) Wenn v nicht mit x_1 beschriftet ist, dann $v[0] = v = v[1]$.

Lemma II: B stellt die Funktion f dar genau dann, wenn $v[0]$ und $v[1]$ die Funktionen $f[0]$ und $f[1]$ darstellen.
Beweis: Einfach.

Satz: Sei $\mathcal{F}$ eine nichtleere Menge von booleschen Funktionen der Arität n und sei $x_{i_1} < \ldots < x_{i_n}$ eine Variablenordnung. Es gibt genau ein multiBDD, das $\mathcal{F}$ darstellt.

Beweis: Wir betrachten die Ordnung $x_1 < x_2 < \ldots < x_n$, für andere ist der Beweis analog.
Beweis durch Induktion über die Arität n.
Basis: $n = 0$. Es gibt zwei boolesche Funktionen mit $n = 0$, nämlich die Konstanten $\mathbf{0}$ und $\mathbf{1}$, und zwei BDDs $\mathbf{K_0}, \mathbf{K_1}$ bestehend aus einem einzigen, mit 0 oder 1 beschrifteten Knoten. $\{\mathbf{0}\}$ wird durch $\mathbf{K_0}$, $\{\mathbf{1}\}$ durch $\mathbf{K_1}$, und $\{\mathbf{0}, \mathbf{1}\}$ durch das multiBDD, das aus $\mathbf{K_0}$ und $\mathbf{K_1}$ besteht, dargestellt.

Schritt: $n > 0$. Sei $\mathcal{F} = \{f_1, \ldots, f_k\}$.
Definiere $\mathcal{F}' = \{f_1[0], f_1[1], \ldots, f_k[0], f_k[1]\}$ mit $f_i[0]$ und $f_i[1]$ wie in Lemma I.
Aus der Induktionsvoraussetzung folgt, dass es genau ein multiBDD B' mit Wurzeln $v_{10}, v_{11}, \ldots, v_{k0}, v_{k1}$ gibt, das $\mathcal{F}'$ darstellt. D.h., für jede Funktion $f_i[j]$ stellt die Wurzel v_{ij} die Funtion $f_i[j]$ dar.

Sei B das multiBDD mit Wurzeln $v_1, \ldots v_n$, der aus B' gewonnen wird, in dem für $i = 1, 2, \ldots, k$ folgendes gemacht wird:

- Wenn $v_{i0} = v_{i1}$, setze $v_i := v_{i0}$.
  (In diesem Fall wird f_i durch v_{i0} dargestellt.)
- Wenn $v_{i0} \neq v_{i1}$ und B' einen Knoten v mit v_{i0} als 0-Kind und v_{i1} als 1-Kind enthält, setze $v_i := v$.
- Wenn $v_{i0} \neq v_{i1}$ und B' keinen solchen Knoten enthält, füge einen neuen Knoten v_i mit v_{i0} als 0-Kind und v_{i1} als 1-Kind zu B hinzu.
  (Damit stellt v_i die Funktion f_i dar, siehe Lemma II.)

B stellt die Menge $\mathcal{F}$ dar. Wir zeigen nun, dass B das einzige multiBDD mit dieser Eigenschaft ist.

Sei $\tilde{B}$ ein beliebiges multiBDD mit Wurzeln $\tilde{v}_1, \ldots \tilde{v}_n$, das $\mathcal{F}$ darstellt.
Aus Lemma II folgt, dass $\tilde{B}$ Knoten $\tilde{v}_1[0], \tilde{v}_1[1], \ldots, \tilde{v}_k[0], \tilde{v}_k[1]$ enthält, die die Funktionen aus $\mathcal{F}'$ darstellen. Aus der Induktionsvoraussetzung folgt, dass das multiBDD, das aus diesen Knoten und ihren Nachfolgern besteht, identisch mit B' ist, d.h., $v_{ij} = \tilde{v}_i[j]$ für alle $i \in \{1, \ldots, k\}$ und $j \in \{0, 1\}$.
Seien v_i und $\tilde{v}_i$ die Wurzeln von B und $\tilde{B}$, die f_i darstellen. Aus Lemma I und II folgt, dass v_{i0} und $\tilde{v}_i[0]$ die Funktion $f[0]$, v_{i1} und $\tilde{v}_i[1]$ die Funktion $f[1]$ darstellen. Da es $v_i[0] = \tilde{v}_i[0]$ und $v_i[1] = \tilde{v}_i[1]$ gilt erhalten wir $v_j = \tilde{v}_j$.
Damit sind B und $\tilde{B}$ identisch.

Berechnung von BDDs aus Formeln

Aufgabe: Gegeben eine Formel F über die atomaren Formeln $A_1, \ldots, A_n$ und eine Variablenordnung für $\{x_1, \ldots, x_n\}$, berechne das BDD, das die Funktion $f_F(x_1, \ldots, x_n)$ darstellt.

Naives Verfahren: Berechne den Entscheidungsbaum von f_F und verkleinere ihn mit Hilfe der Komprimierungsregeln.

Problem: Der Entscheidungsbaum ist viel zu groß!

Besseres Verfahren (Idee): Berechne rekursiv das multiBDD für $\{f_{F[A_i/0]}, f_{F[A_i/1]}\}$ für ein geeignetes A_i, und leite daraus das BDD für f_F ab.
Hierbei ist $F[A_i/0]$ bzw. $F[A_i/0]$ die Formel, die man erhält, in dem jedes Vorkommenvon A_i durch 0 bzw. durch 1 ersetzt.
In den folgenden Folien bearbeiten wir diese Idee.

Sei $\mathcal{M} = \{F_1, \ldots, F_n\}$ eine nichtleere Menge von Formeln.

Wir definieren eine Prozedur multiBDD($\mathcal{M}$), die die Wurzeln eines multiBDD zurückgibt, das die Menge $\{f_{F_1}, \ldots, f_{F_n}\}$ darstellt.

$\mathbf{K_0}$ bezeichnet den mit 0 beschrifteten Knoten.
$\mathbf{K_1}$ bezeichnet den mit 1 beschrifteten Knoten.

Eine echte Formel ist eine Formel, in der mindestens eine Variable vorkommt (d.h. nicht nur 0 und 1).

Eine atomare Formel A_i ist kleiner als A_j, wenn x_i vor x_j in der Variablenordnung vorkommt.

Die Funktion multiBDD($\mathcal{M}$)

if $\mathcal{M}$ keine echte Formel enthält
then if alle Formeln in $\mathcal{M}$ äquivalent zu 0 sind
    then return $\{\mathbf{K_0}\}$
    else if alle Formeln in $\mathcal{M}$ äquivalent zu 1 sind
    then return $\{\mathbf{K_1}\}$
    else return $\{\mathbf{K_0}, \mathbf{K_1}\}$
else Wähle eine echte Formel $F \in \mathcal{M}$.
    Sei A_i die kleinste atomare Formel, die in F vorkommt.
    Sei $B =$ multiBDD($(\mathcal{M} \setminus \{F\}) \cup \{F[A_i/0], F[A_i/1]\}$).
    Seien v_0, v_1 die Wurzeln von B, die $F[A_i/0], F[A_i/1]$ darstellen.
    if $v_0 = v_1$ **then return** B
    else füge einen Knoten v mit v_0, v_1 als 0- und 1-Kind
        (falls es keinen solchen Knoten schon gibt);
        return $(B \setminus \{v_0, v_1\}) \cup \{v\}$

Äquivalenzprobleme

Gegeben zwei Formeln F_1, F_2, das folgende Verfahren entscheidet, ob $F_1 \equiv F_2$ gilt:

- Wähle eine geeignete Variablenordnung $x_1 < \ldots < x_n$.
- Berechne ein multiBDD für $\{F_1, F_2\}$.
- Prüfe, ob die Wurzel v_{F_1}, v_{F_2} identisch sind.

Für Schaltkreise: die BDDs werden nicht aus Formeln, sondern direkt aus dem Schaltkreis generiert.

Operationen mit BDDs

Gegeben:

- zwei Formel F, G über die atomaren Formeln $A_1, \ldots, A_n$,
- eine Variablenordnung für $\{x_1, \ldots, x_n\}$,
- ein multiBDD mit zwei Wurzeln v_F, v_G, die die Funktionen $f_F(x_1, \ldots, x_n)$ und $f_F(x_1, \ldots, x_n)$ darstellen, und
- eine binäre Operation (e.g. $\vee, \wedge, \rightarrow, \leftrightarrow$)

Aufgabe: berechne ein BDD für die Funktion $f_{F \circ G}(x_1, \ldots, x_n)$.

Mit unserer Konvention gilt $f_{F \circ G} = f_F \circ f_G$

Idee des Verfahrens

Lemma: $(f_F \circ f_G)[0] = f_F[0] \circ f_G[0]$ und $(f_F \circ f_G)[1] = f_F[1] \circ f_G[1]$.
Beweis: Aufgabe.

Verfahren: (für die Ordnung $x_1 < x_2 < \ldots < x_n$, für andere analog)

- Berechne ein multiBDD für $\{f_F[0] \circ f_G[0], f_F[1] \circ f_G[1]\}$. (Rekursiv.)
- Baue mit Hilfe des Lemmas ein BDD für $f_{F \circ G}(x_1, \ldots, x_n)$.

Die Funktion Oder(v_F, v_G)

if $v_F = \mathbf{K_1}$ oder $v_F = \mathbf{K_1}$ **then return** $\mathbf{K_1}$
else if $v_F = v_G = \mathbf{K_0}$ **then return** $\mathbf{K_0}$
else Seien v_{F0}, v_{G0} die 0-Kinder von v_F, v_G und
    seien v_{F1}, v_{G1} die 1-Kinder von v_F, v_G
    $v_0 :=$ Oder(v_{F0}, v_{G0}); $v_1 :=$ Oder(v_{F1}, v_{G1})
    if $v_0 = v_1$ **then return** v_0
    else füge einen Knoten v mit v_0, v_1 als 0- und 1-Kind
        (falls es keinen solchen Knoten schon gibt);
        return v

Eine Implementierung von BDD

Quelle: *An introduction to Binary Decision Diagrams*

Prof. H.R. Andersen

http://www.itu.dk/people/hra/notes-index.html

Datenstrukturen

BDD-Knoten kodiert als Zahlen $0, 1, 2, \ldots$ mit 0, 1 für die Endknoten

BDD-Knoten werden in einer Tabelle

$$T : u \mapsto (i, l, h)$$

gespeichert, wobei i, l, h die Variable, das 0-Kind und das 1-Kind von u sind. (Hier steht l für "low" und h für "high".)

Wir verwalten auch eine zweite Tabelle

$$H : (i, l, h) \mapsto u$$

so daß die folgende Invariante gilt:

$$T(u) = (i, l, h) \quad \text{iff} \quad H(i, l, h) = u$$

Basis-Operationen auf T:

$init(T)$:	Initialisiert T mit 0 und 1
$add(T, i, l, h)$:	Gibt Knoten mit Attributen (i, l, h) zurück
$var(u)$, $low(u)$, $high(u)$:	Gibt die Variable, das 0- oder das 1-Kind von u zurück

Basis-Operationen auf H:

$init(H)$:	Initialisiert H als die leere Tabelle
$member(H, i, l, h)$:	Prüft, ob (i, l, h) in H liegt
$lookup(H, i, l, h)$:	Gibt den Knoten $H(i, l, h)$ zurück
$insert(H, i, l, h, u)$:	fügt $(i, l, h) \mapsto u$ zu H (wenn noch nicht da)

Die Funktion $Make(i, l, h)$

Suche in H nach einem Knoten mit Attributen (i, l, h). Wenn es den Knoten gibt, dann gib ihn zurück. Sonst erzeuge einen neuen Knoten und gib ihn zurück.

$Make(i, l, h)$

1: **if** $l = h$ **then return** l
2: **else if** $member(H, i, l, h)$ **then**
3: **return** $lookup(H, i, h, l)$
4: **else** $u := add(T, i, l, h)$
5: $insert(H, i, l, h, u)$
6: **return** u

Implementierung von Oder

Problem: die Funktion kann mehrmals mit denselben Argumenten aufgerufen werden!

Lösung: Dynamische Programmierung. Die Ergebnisse sämtlicher Aufrufe werden gespeichert. Bei jedem Aufruf wird erst geprüft, ob das Ergebnis schon berechnet wurde.

$Oder(u_1, u_2)$
1: **init** G
2: return $Oder'(u_1, u_2)$

$Oder'(u_1, u_2)$

1: **if** $G(u_1, u_2) \neq empty$ **then return** $G(u_1, u_2)$
2: **else if** $u_1 = 1$ **or** $u_2 = 1$ **then return** 1
3: **else if** $u_1 = 0$ **and** $u_2 = 0$ **then return** 0
4: **else if** $var(u_1) = var(u_2)$ **then**
5: $u := Make(var(u_1), Oder'(low(u_1), low(u_2)), Oder'(high(u_1), high(u_2)))$
6: **else if** $var(u_1) < var(u_2)$ **then**
7: $u := Make(var(u_1), Oder'(low(u_1), u_2), Oder'(high(u_1), u_2))$
8: **else** $u := Make(var(u_2), Oder'(u_1, low(u_2)), Oder'(u_1, high(u_2)))$
9: $G(u_1, u_2) = u$
10: **return** u