Algorithmen und Datenstrukturen

Praktische Einführung und Programmierung

Stefan Bosse

Universität Koblenz - FB Informatik

Binäre Entscheidungsbäume

Ausflug in die Digitallogik und Boolesche Algebra

- Digitallogikschaltungen sind einerseits die Grunglage um unsere Algorithmen ausführen zu können
- Aber Digitallogikschaltungen haben auch etwas mit Graphen und Bäumen zu tun:

1. Eine Digitallogikschaltung sowie jede Elektronikschlatung ist ein Netzwerk aus Bauelementen die miteinander verbunden sind ⇒ **nicht gerichteter zyklischer Graph**
2. Eine Digitallogikschaltung kann durch Boolesche Algebra, Boolesche Funktionen und Wahrheitstabellen mathematisch modelliert und behandelt werden.
3. Ein Boolesche Funktion kann in einen Baum entwickelt (transformiert) werden!

Digitallogik

Digitallogik kann auf verschiedenen Abstraktionsebenen betrachtet werden:

1. Transistorschaltungen (oder noch tiefer Mikrochip / physikalisch)
2. Gatterebene: Und-, Oder-Verknüpfung, Inverter, Kombinationen daraus
3. Blockebene: Addierer, Register, Shieberegister, Multiplexer
4. Boolesche Algebra: **Boolesche Funktion** (aber nur "zeitlose" kombinatorische Logik)
5. Boolesche Algebra: **Entscheidungsbäume** oder **Entscheidundsdiagramme**

Digitallogik

[bddrued]

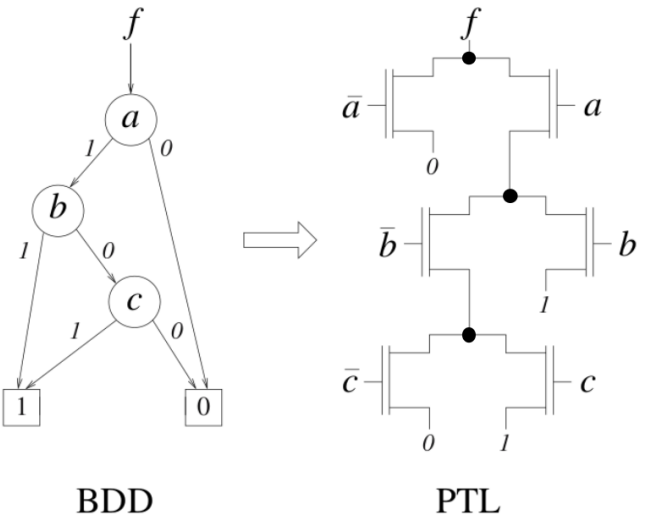


Abb. 1. Bäume können auch direkt als Elektronikschaltung implementiert werden!

Boolesche Funktionen

Eine Boolesche Funktion besteht aus:

1. Konstanten, Werten 0/1/False/TRue
2. Variablen (Eingänge einer Digitallogikschaltung)
3. Termen
4. Verknüpfungen: Konjunktion (Und), Disjunktion (Oder), Negation

Eine Boolesche Funktione bildet einen Eingangsvektor $\boldsymbol{x}$ auf einen skalaren Wert y ab:

$$f(\vec{x}) = \vec{x} \rightarrow y : \mathbb{B}^n \rightarrow \mathbb{B}$$
$$F(\vec{x}) = \vec{x} \rightarrow \vec{y} : \mathbb{B}^n \rightarrow \mathbb{B}^m \Leftrightarrow$$
$$F(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), .., f_m(\vec{x}))$$

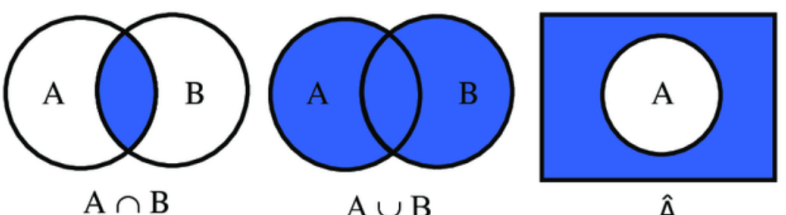
Hat eine Digitallogikschaltung mehr als einen Ausgang y so können diese durch getrennte Boolesche Funktionen berechntet werden.

Boolesche Verknüpfungen

Es gibt drei grundlegende Boolesche Operatoren die Boolesche Werte und Variablen zu Termen verknüpfen:

$$\text{Konjunktion} = \text{Und} \Leftrightarrow a \wedge b \wedge c \wedge \ldots \Leftrightarrow a \cdot b \cdot c \cdot \ldots$$
$$\text{Disjunktion} = \text{Oder} \Leftrightarrow a \vee b \vee c \vee \Leftrightarrow a + b + c + \ldots$$
$$\text{Negation} : \neg x \Leftrightarrow \overline{x}$$

a	b	a ∧b	a ∨ b	¬ a
0	0	0	0	1
1	0	0	1	0
0	1	0	1	1
1	1	1	1	0



Zerlegung Boolescher Funktionen

Man kann jeder Boolesche Funktion nach ihren variablen wie folgt zerlegen:

$$f = x_i \cdot f_{x_i=1} + \overline{x}_i \cdot f_{x_i=0}$$

Dabei werden Variablen x einzeln mit den Werten 0 und 1 festgelegt und erzeugen dabei zwei neue Teilfunktionen $f_{x=0}$ und $f_{x=1}$

Beispiel:

$$\begin{aligned} f &= a \cdot b + c \\ &= a \cdot (b + c) + \overline{a} \cdot (c) \\ &= b \cdot \left(a + \overline{a} \cdot c\right) + \overline{b} \cdot \left(a \cdot c + \overline{a} \cdot c\right) \end{aligned}$$

Wahrheitstabellen

Jede Funktion, jede Boolesche Funktion kann als eine Tabelle wenigstens partiell dargestellt werden.

- Es gilt die Äquivalenz:

```
Funktion ⇔ Tabelle ⇔ Evaluierungsbaum
```

Funktion

$$f(a,b) = y = a \wedge b$$

Funktionstabelle

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

Binäre Entscheidungsbäume

Können Bäume oder Graphen sein!! Besser Entscheidungsdiagramme!

Zur Erinnerung: Ein Baum hat gerichtete Kanten und jeder Knoten hat maximal einen Vorgänger. Ein Graph hat diese Einschränkungen nicht.

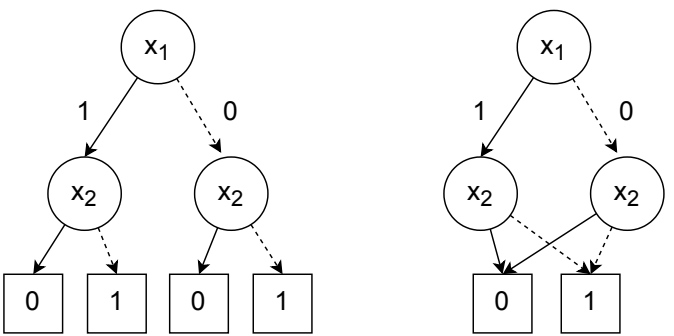


Abb. 2. (Links) Echter Binärer Entscheidungsbaum (Rechts) Äquivalenter Graph

Binäre Entscheidungsbäume

Ein BDD is ein Evaluierungsbaum für Boolesche Funktionen (oder Suche von y!) und besteht aus:

1. Knoten n_i die mit Variablen x_j der Booleschen Funktion $f(x_1,x_2,..)$ verknüpft sind.
2. Jeder Knoten $n_i(x_j)$ hat zwei ausgehende Kanten (1) und (0), jeweils eine für x_j=1 und eine (gestrichelt) für x_j=0.
3. Jeder Knoten hat mindestens einen Vorgänger außer dem Wurzelknoten
4. Jeder Ast im Baum ist wieder ein Baum
5. Es gibt nut zwei Arten von Blättern (Terminalknoten): (1) und (0). Diese liefern das Ergebnis der Booleschen Funktion y
6. Allgemein ist ein BDD ein Gerichteter azyklischer Graph (DAG)

Um eine Boolesche Funktion die durch einen BDD gegeben ist zu berechnen fängt man im Wurzelknoten an und folget den Knoten und Verbindungen gemäß konkreter Werte für x_1, x_2 usw. bis ein Blattknoten erreicht ist.

Binäre Entscheidungsbäume

Evaluierung

```
function Node(xi) { return { type:'Node', left:null, right:null, xindex:xi }  }
function Leave(v) { return { type:'Leave', value:v } }
function evalBDD(root,xv) {
  function iter(node) {
    if (node.type == 'Leave') return node.value // we are done
    if (xv[node.xindex]==1) return iter(node.left)
    else                    return iter(node.right)
  }
  return iter(root)
}
// f(x1=0,x2=1,x3=1)
y=evalBDD(root,[0,1,1])
```

Alg. 1. Evaluierung eines BDD mit 0/1 als Ergebnis

Erzeugung

Wie kann nun eine Boolesche Funktion in einen Baum transformiert werden?

Die Probleme:

- Es gibt unendlich viele Bäume die einer Booleschen Funktion f äquivalent sind, da:
- Es gibt unendlich viele Boolesche Funktionen $f_1, f_2, ...$ die einer Booleschen Funktion f äquivalent sind
- Die Erzeugung von vollständig geordneten und reduzierten (optimalen) BDD (ROBDD) ist ein NP-vollständiges Problem!

Lösung 1: Shanon Zerlegung wobei jeweils ein Knoten x_i geschaffen wird und die reusltierenden Funktionen den linken und rechten Teilbaum erzeugen werden.

$$f = x_i \cdot f_{x_i=1} + \overline{x}_i \cdot f_{x_i=0}$$

Erzeugung

Lösung 2: Direkt aus einer Wahrheitstabelle

a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

a=1

b	c	y
0	0	1
0	1	0
1	0	1
1	1	0

a=0

b	c	y
0	0	0
0	1	0
1	0	1
1	1	1

b=1 (a=1)

c	y
0	1
1	0

b=1 (a=0)

c	y
0	1
1	1

b=0 (a=1)

c	y
0	1
1	0

b=0 (a=0)

c	y
0	0
1	0

Erzeugung

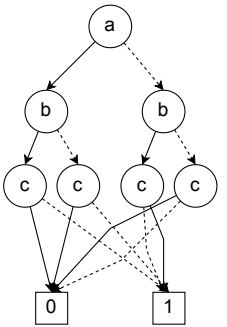


Abb. 3. BDD Erzeugeung aus Wahrheitstabelle

- Das Gute zuerst: Leiten wir den Baum aus der Wahrheitstabelle durch schtittweise Reduzierung der Zeilen und Spalten (Arrays) mit gordneter Variablenreihenfolge ab dann ist der Baum symmetrisch, vollständig balanziert, und geordnert (Rechenkomplexität der Evaluierung ist O(n) bei n Variablen, die Höhe ist auch n)
- Das Schleche danach: Der Baum ist überbestimmt und kann reduziert werden.

Erzeugung

[bddrued]

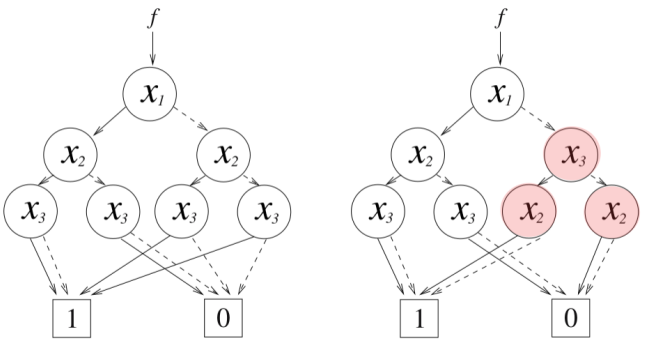


Abb. 4. (Links) Geordneter Baum (Rechts) Nicht geordneter Baum

Die Reihenfolge der Zerlegung nach Variablen (Reihenfolge der Variablen) bestimmt die Baumstruktur! Bei einer bereits teilreduzierten Booleschen Funktion ist Ergebnis nicht vorhersagbar (auch die Höhe).

Erzeugung

```
function Node(xi) { return { type:'Node', left:null, right:null, xindex:xi }  }
function Leave(v) { return { type:'Leave', value:v } }
// [a,b,c,y]
tt = [[ 0, 0, 0, 0], [ 0, 0, 1, 0], [ 0, 1, 0, 1], [ 0, 1, 1, 1],
      [ 1, 0, 0, 1], [ 1, 0, 1, 0], [ 1, 1, 0, 1], [ 1, 1, 1, 0]]
function split(numvar,index,tt) {
  if (index==numvar-1) {
    node = Node(index)
    node.left  = Leave(tt[0][1])
    node.right = Leave(tt[1][1])
  } else {
    tt1 = map(filter(tt,(row => row[0]==1)),row => tail(row))
    tt0 = map(filter(tt,(row => row[0]==0)),row => tail(row))
    node = Node(index)
    node.left  = split(numvar,index+1,tt1)
    node.right = split(numvar,index+1,tt0)
  }
  return node
}
root = split(3,0,tt)
```

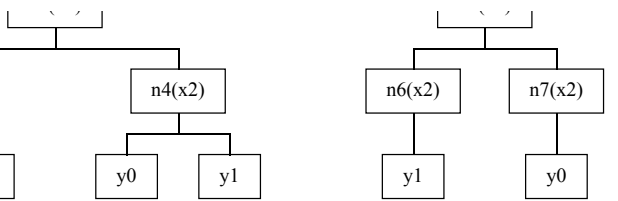
Alg. 2. Rekursive Erzeugung eines BDD aus Wahrheitstabelle

BDD Live

+ Ø Ⓟ ▶ bdd1.js bdd2.js bdd3.js bdd4.js ?

```
1 tt = [[ 0, 0, 0, 0], [ 0, 0, 1, 0], [ 0, 1, 0, 1], [ 0, 1, 1, 1],
2       [ 1, 0, 0, 1], [ 1, 0, 1, 0], [ 1, 1, 0, 1], [ 1, 1, 1, 0]]
3 root = split(3,0,tt)
4 treeview(toView(root,0))
5 ttTest = map(tt,row =>
6   concat(slice(row,0,2),
7          [evalBDD(root,slice(row,0,2))]))
8 print(ttTest)
```

```
[ [ 0, 0, 0, 0 ],
  [ 0, 0, 1, 0 ],
  [ 0, 1, 0, 1 ],
  [ 0, 1, 1, 1 ],
  [ 1, 0, 0, 1 ],
  [ 1, 0, 1, 0 ],
  [ 1, 1, 0, 1 ],
  [ 1, 1, 1, 0 ] ]
```



+

Reduktion von BDD

Ziel ist die Verringerung der Knoten im BDD und somit Terme der Booleschen Funktion.

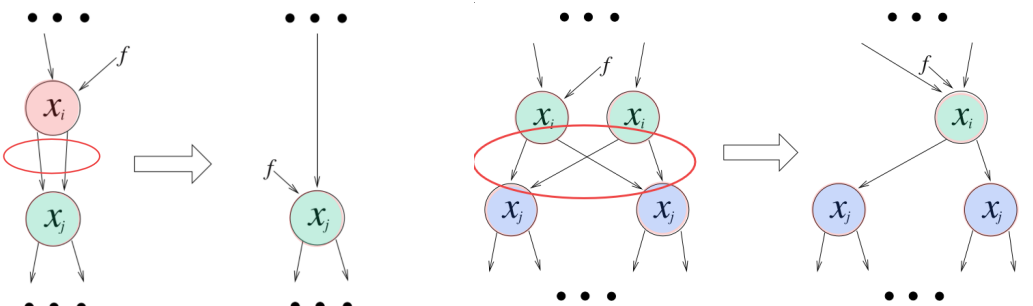


Abb. 5. (Links) Transformatin 1: Löschen (Rechts) Transformation 2: Zusammenführung

Reduktion von BDD

1. Bei der Löschregel gibt es einen Knoten x_i dessen beide Kanten den gleichen Nachfolgerknoten x_j besitzen. Der Knoten x_i kann entfernt werden. Alle eingehenden Kanten werden auf x_j umgelegt.

2. Bei der Zusammenführungsregel gibt es vier Knoten mit zweimal x_i und zweimal x_j mit jeweils gekreuzten Kanten von den x_i zu den x_j Knoten. Die beiden x_i Knoten können zusammengelegt werden. Alle eingehenden Kanten werden auf den zusammengelegten Knoten mit x_i umgelegt.

Reduktion von BDD

Der Vorgang ist iterativ und bottom-up.

- Man beginnt bei den Blättern und arbeitet sich bis zur Wurzel hoch.