

Kapitel 5

Aussagenlogik

Aussagen

Unter einer **Aussage** verstehen wir einen Satz, der entweder wahr oder falsch ist.

Wir müssen nicht wissen, ob der Satz wahr oder falsch ist, wir müssen ihm nur prinzipiell zugestehen, dass er wahr oder falsch ist. Genau genommen interessieren wir uns nicht einmal dafür, ob der Satz wahr oder falsch ist. Und genau genommen interessieren wir uns nicht einmal dafür, was "wahr" und "falsch" inhaltlich bedeutet. Wir können jederzeit 0 oder 1 statt "falsch" oder "wahr" sagen. Insofern betreiben wir Logik als ein rein formales System ohne Bezug zur Realität.

Konkrete Aussagen sind z. B.:

Köln liegt in Deutschland.

Köln hat mehr als 1 Mio. Einwohner.

Keine Aussagen im Sinne unserer Begriffsbildung sind dagegen:

Guten Tag, meine Damen und Herren!

Wie spät ist es?

Bei der Programmierung haben wir es nicht mit umgangssprachlichen Aussagen, sondern mit präzise formulierten Aussagen in einer Programmiersprache wie `"wert == 10"` oder `"a+b < c"` zu tun.

Aussagenlogische Operatoren

Für die Wahrheitswerte "wahr" bzw. "falsch" benutzen wir die Symbole 1 bzw. 0. Für Aussagen setzen wir Großbuchstaben, also A, B oder C. Die Aussage "A ist wahr" heißt dann in Formelschreibweise " $A = 1$ ". Umgekehrt heißt " $A = 0$ ": "Die Aussage A ist falsch".

Aussagen verknüpfen wir durch logische Operatoren wie "nicht", "und" oder "oder". Diese Operatoren können wir durch sogenannte Wahrheitstabellen definieren:

A	nicht A
0	1
1	0

A	B	A und B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A oder B
0	0	0
0	1	1
1	0	1
1	1	1

Beachten Sie, dass der Operator "oder" nicht ausschließend verwendet wird. Das heißt, eine mit "oder" zusammengesetzte Aussage ist wahr, wenn mindestens eine der beiden Teilaussagen wahr ist.

Boolesche Ausdrücke

Operator	Notation in Formeln	Notation in C	Priorität
nicht A	\bar{A}	!	1 (hoch)
A und B	$A \wedge B$	&&	2
A oder B	$A \vee B$		3 (niedrig)

Mit booleschen Variablen, Operatoren und Klammern können boolesche Ausdrücke gebildet werden:

$$(\bar{A} \vee B) \wedge \overline{(C \vee A)}$$

In C-Notation:

$$(!A \ || \ B) \ \&\& \ !(C \ || \ A)$$

Prioritäten dienen dazu, Klammern zu sparen. $A \&\&B \ || \ C$ entspricht $(A \&\&B) \ || \ C$.

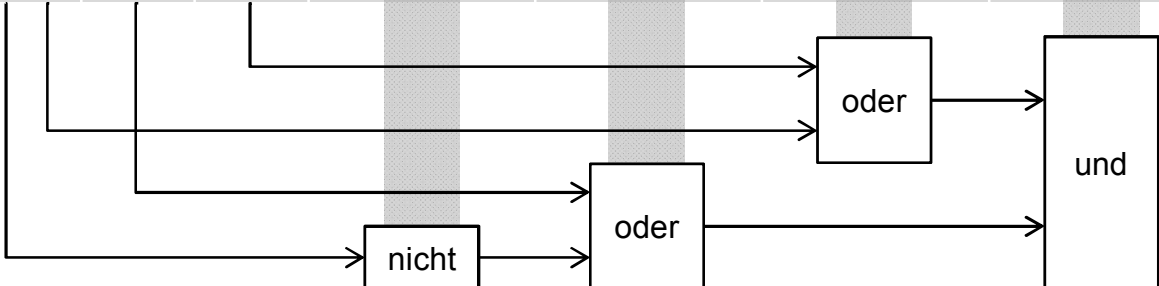
Im Zweifel oder wenn eine bestimmte Auswertung gewünscht ist, setzen Sie Klammern:

$$A \&\& (B \ || \ C)$$

Wahrheitstabellen

Die Wahrheitswerte boolescher Ausdrücke wie zum Beispiel $(\bar{A} \vee B) \wedge (C \vee A)$ können mit Wahrheitstabellen ermittelt werden:

A	B	C	\bar{A}	$\bar{A} \vee B$	$C \vee A$	$(\bar{A} \vee B) \wedge (C \vee A)$
0	0	0	1	1	0	0
0	0	1	1	1	1	1
0	1	0	1	1	0	0
0	1	1	1	1	1	1
1	0	0	0	0	1	0
1	0	1	0	0	1	0
1	1	0	0	1	1	1
1	1	1	0	1	1	1



Äquivalenzen

Verschiedene boolesche Ausdrücke können für alle Belegungen der eingehenden Variablen die gleichen Wahrheitswerte haben.

Solche Ausdrücke bezeichnen wir als äquivalent oder logisch gleichwertig.

Mit dem Äquivalenzoperator "genau dann wenn", dem wir eine niedrigere Priorität als allen bisher eingeführten Operatoren geben, ergibt sich dann ein, unabhängig von den eingehenden booleschen Variablen, immer wahrer Ausdruck:

$$\overline{A \wedge B} \Leftrightarrow \overline{A} \vee \overline{B}$$

Der Äquivalenzoperator ist das Gleichheitszeichen der Aussagenlogik.

A	B	$\overline{A \wedge B}$	$\overline{A} \vee \overline{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Die Spalten sind gleich

A	B	$A \Leftrightarrow B$
0	0	1
0	1	0
1	0	0
1	1	1

A	B	$\overline{A \wedge B}$	$\overline{A} \vee \overline{B}$	$\overline{A \wedge B} \Leftrightarrow \overline{A} \vee \overline{B}$
0	0	1	1	1
0	1	1	1	1
1	0	1	1	1
1	1	0	0	1

Dieser Ausdruck ist immer wahr

Wichtige Äquivalenzen

Logische Äquivalenzen		
$A \wedge (B \wedge C) \Leftrightarrow (A \wedge B) \wedge C$	$A \vee (B \vee C) \Leftrightarrow (A \vee B) \vee C$	Assoziativgesetz
$A \wedge B \Leftrightarrow B \wedge A$	$A \vee B \Leftrightarrow B \vee A$	Kommutativgesetz
$(A \vee B) \wedge A \Leftrightarrow A$	$(A \wedge B) \vee A \Leftrightarrow A$	Verschmelzungsgesetz
$A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$	$A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$	Distributivgesetz
$A \wedge (B \vee \overline{B}) \Leftrightarrow A$	$A \vee (B \wedge \overline{B}) \Leftrightarrow A$	Komplementgesetz
$A \wedge A \Leftrightarrow A$	$A \vee A \Leftrightarrow A$	Idempotenzgesetz
$\overline{A \wedge B} \Leftrightarrow \overline{A} \vee \overline{B}$	$\overline{A \vee B} \Leftrightarrow \overline{A} \wedge \overline{B}$	De Morgansches Gesetz
$A \wedge \overline{A} \Leftrightarrow 0$	$A \vee \overline{A} \Leftrightarrow 1$	
$\overline{\overline{A}} \Leftrightarrow A$		

Die Assoziativgesetze ermöglichen klammerfreie Schreibweise wie $A \wedge B \wedge C$ oder $A \vee B \vee C$.

Zur besseren Lesbarkeit lässt man das \wedge – Zeichen in Formeln häufig weg.

Äquivalenzen können verwendet werden, um boolesche Formeln ohne Wertänderung umzuformen, um sie etwa zu vereinfachen.

$$\begin{aligned}
 & AB\overline{C}\overline{D} \vee AB\overline{C}D \vee ABC\overline{D} \vee ABCD \\
 & \Leftrightarrow AB(\overline{C}\overline{D} \vee \overline{C}D \vee C\overline{D} \vee CD) \\
 & \Leftrightarrow AB(\overline{C}(\overline{D} \vee D) \vee C(\overline{D} \vee D)) \\
 & \Leftrightarrow AB(\overline{C} \vee C) \\
 & \Leftrightarrow AB
 \end{aligned}$$

Der Implikationsoperator

Der Implikationsoperator drückt eine "wenn-dann" Beziehung zwischen seinen Operanden aus:

A	B	$A \Rightarrow B$
0	0	1
0	1	1
1	0	0
1	1	1

Dem Implikationsoperator geben wir eine höhere Priorität als dem Äquivalenzoperator, aber eine niedrigere Priorität als "nicht", "und" und "oder". Beachten Sie, dass die Implikation immer wahr ist, wenn die Prämisse (hier A) falsch ist.

Regeln über Äquivalenz und Implikation
$(A \Rightarrow B) \Leftrightarrow (\overline{A} \vee B)$
$(A \Rightarrow B) \Leftrightarrow (\overline{B} \Rightarrow \overline{A})$
$(A \Leftrightarrow B) \Leftrightarrow (A \Rightarrow B) \wedge (B \Rightarrow A)$

Für Implikation und Äquivalenz gibt es keine Operatoren in der Programmiersprache C

Statt einer Implikation $A \Rightarrow B$ verwendet man `!A || B`.

Anstelle einer Äquivalenz kann man einfach den Test auf Gleichheit (`==`) verwenden.

Umgangssprachliche Verwendung logischer Operatoren

Die Qualität der Aussagenlogik liegt in der "Algebraisierung" der Logik, das heißt, in der Reduktion auf genau definierte Operatoren und einen Kalkül, der es ermöglicht, mit logischen Ausdrücken wie mit mathematischen Formeln zu rechnen. Damit entfernt sich die formale Logik von der "Alltagslogik".

Wenn wir umgangssprachlich "oder" verwenden, meinen wir häufig "entweder oder". Auf die Frage "Bier oder Wein?" erwarten wir nicht die Antwort "beides". Trotzdem verwenden wir das nicht ausschließende "oder" ganz selbstverständlich auch in der Umgangssprache. Wenn Sie zum Beispiel an der Grenze von einem Zöllner gefragt werden, ob Sie den Pass oder den Personalausweis dabei haben, würden Sie dann mit nein antworten, wenn Sie zufällig beide Dokumente dabei haben? Ob "oder" ausschließend oder nicht ausschließend gemeint ist, erkennt man oft nur aus dem Zusammenhang.

Wenn wir umgangssprachlich "wenn dann" verwenden, meinen wir häufig "genau dann wenn". Der Satz: "Wenn morgen die Sonne scheint, dann gehe ich ins Schwimmbad" wird gemeinhin so verstanden, dass man, wenn die Sonne nicht scheint, auch nicht ins Schwimmbad geht. Streng logisch ist das aber nicht so. Ich kann bei Regen ins Schwimmbad gehen, ohne in Widerspruch zu meiner Aussage zu kommen, weil ich mich für den Fall, dass die Sonne nicht scheint, nicht festgelegt habe.

Was die Programmierung betrifft, darf es solche unscharfen, nur aus dem Zusammenhang zu verstehenden Aussagen nicht geben. Halten Sie sich daher immer an die hier definierten Operatoren mit den hier definierten Regeln.

Boolesche Funktionen

Mit booleschen Funktionen kann man beliebige funktionale Abhängigkeiten eines booleschen Wertes von einer oder mehreren booleschen Variablen modellieren:

Hier stehen alle möglichen
Wertkombinationen für die
Eingabeparameter

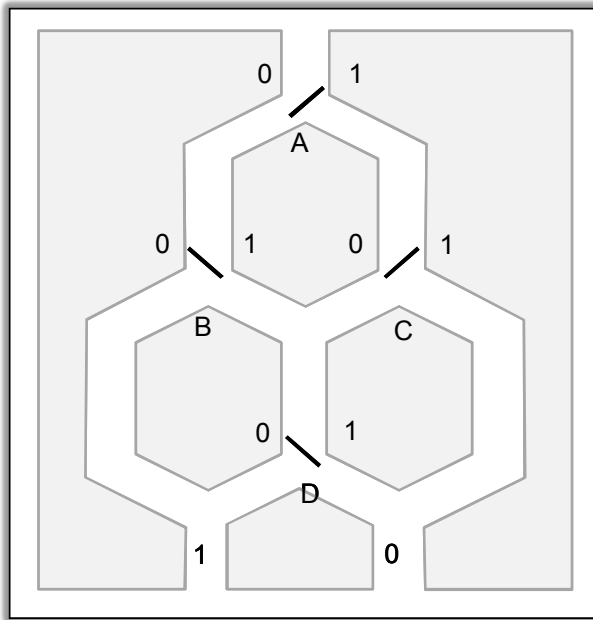
a_1	a_2	a_3	a_4	$z = f(a_1, a_2, a_3, a_4)$
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Dies ist eine 4-stellige
boolesche Funktion

Dies ist ein
Funktionsergebnis
 $z = f(0, 1, 1, 0)$

Jeder boolesche Ausdruck ist eine boolesche Funktion, aber gibt es auch zu jeder booleschen Funktion einen booleschen Ausdruck?

Ein Beispiel für eine Boolesche Funktion



Das Beispiel zeigt, dass man jede boolesche Funktion als booleschen Ausdruck mit "nicht", "und" und "oder" darstellen kann.

Darstellung als Ausdruck:

$$z = \bar{A}\bar{B}AC\wedge D \vee \bar{A}BAC\wedge D \vee A\bar{B}\bar{A}\bar{C}\wedge D \vee A\bar{B}AC\wedge D \vee A\bar{B}\bar{A}\bar{C}\wedge \bar{D} \vee A\bar{B}AC\wedge \bar{D} \vee A\bar{B}AC\wedge \bar{D} \vee A\bar{B}AC\wedge D$$

Ohne und-Zeichen:

$$z = \bar{A}\bar{B}CD \vee \bar{A}BCD \vee A\bar{B}\bar{C}\bar{D} \vee A\bar{B}CD \vee AB\bar{C}\bar{D} \vee AB\bar{C}D \vee ABC\bar{D} \vee ABCD$$

A	B	C	D	$z = f(A, B, C, D)$	
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	1	$\bar{A}\bar{B}AC\wedge D$
0	1	0	0	0	
0	1	0	1	0	
0	1	1	0	0	
0	1	1	1	1	$\bar{A}BAC\wedge D$
1	0	0	0	0	
1	0	0	1	1	$A\bar{B}\bar{A}\bar{C}\wedge D$
1	0	1	0	0	
1	0	1	1	1	$A\bar{B}AC\wedge D$
1	1	0	0	1	$A\bar{B}\bar{A}\bar{C}\wedge \bar{D}$
1	1	0	1	1	$A\bar{B}AC\wedge \bar{D}$
1	1	1	0	1	$A\bar{B}AC\wedge \bar{D}$
1	1	1	1	1	$A\bar{B}AC\wedge D$

Vereinfachung boolescher Ausdrücke durch Äquivalenzen

$$z = \bar{A}\bar{B}CD \vee \bar{A}BCD \vee A\bar{B}\bar{C}D \vee A\bar{B}CD \vee AB\bar{C}\bar{D} \vee AB\bar{C}D \vee ABC\bar{D} \vee ABCD$$

$$\Leftrightarrow \bar{A}\bar{B}CD \vee \bar{A}BCD \vee A\bar{B}\bar{C}D \vee A\bar{B}CD \vee AB(\bar{C}\bar{D} \vee \bar{C}D \vee C\bar{D} \vee CD)$$

$$\Leftrightarrow \bar{A}\bar{B}CD \vee \bar{A}BCD \vee A\bar{B}\bar{C}D \vee A\bar{B}CD \vee AB(\bar{C}(\bar{D} \vee D) \vee C(\bar{D} \vee D))$$

$$\Leftrightarrow \bar{A}\bar{B}CD \vee \bar{A}BCD \vee A\bar{B}\bar{C}D \vee A\bar{B}CD \vee AB(\bar{C} \vee C)$$

$$\Leftrightarrow \bar{A}\bar{B}CD \vee \bar{A}BCD \vee A\bar{B}\bar{C}D \vee A\bar{B}CD \vee AB$$

$$\Leftrightarrow (\bar{A}\bar{B}C \vee \bar{A}BC \vee A\bar{B}\bar{C} \vee A\bar{B}C)D \vee AB$$

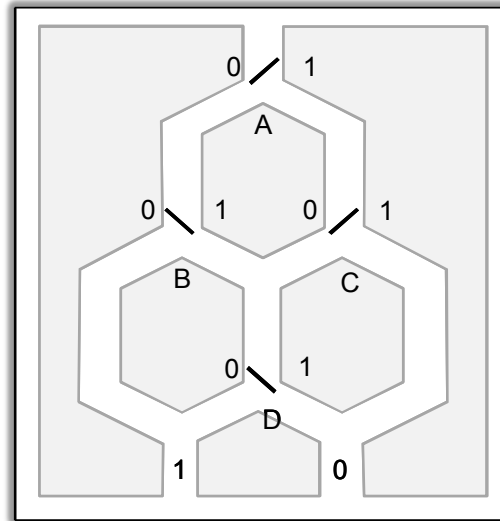
$$\Leftrightarrow (\bar{A}C(\bar{B} \vee B) \vee A\bar{B}(\bar{C} \vee C))D \vee AB$$

$$\Leftrightarrow (\bar{A}C \vee A\bar{B})D \vee AB$$

Also: $z = AB \vee (\bar{A}\bar{B} \vee \bar{A}C)D$

Systematische Vereinfachung durch Karnaugh-Diagramme (\rightarrow Übungen).

Ad hoc Vereinfachung



Wenn $A=1$ und $B=1$ ist, werden alle Kugeln zum Ausgang 1 gelenkt, egal wie die beiden anderen Weichen stehen.

Wenn $A=1$ und $B=0$ ist oder wenn $A=0$ und $C=1$ ist, geht die Kugel durch die Mitte und D entscheidet, wo sie letztlich hingeht.

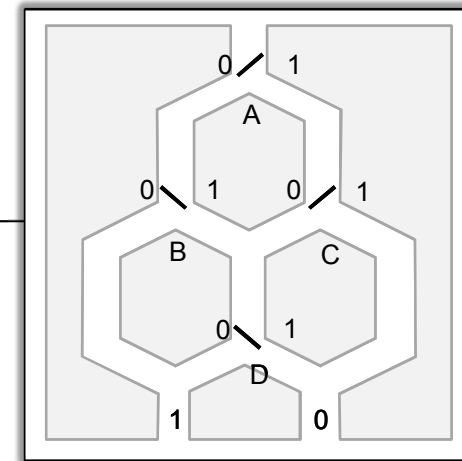
In allen anderen Fällen ist der Ausgang 0.

$$\text{Also: } z = AB \vee (\overline{A}\overline{B} \vee \overline{A}C)D$$

Programmierbeispiel

```
void main()
{
    int A, B, C, D;
    int z;

    for( A = 0; A <= 1 ; A++)
    {
        for( B = 0; B <= 1 ; B++)
        {
            for( C = 0; C <= 1 ; C++)
            {
                for( D = 0; D <= 1 ; D++)
                {
                    z = A&&B || (A&&!B || !A&&C) && D;
                    printf( "%d %d %d %d | %d\n", A, B, C, D, z);
                }
            }
        }
    }
}
```

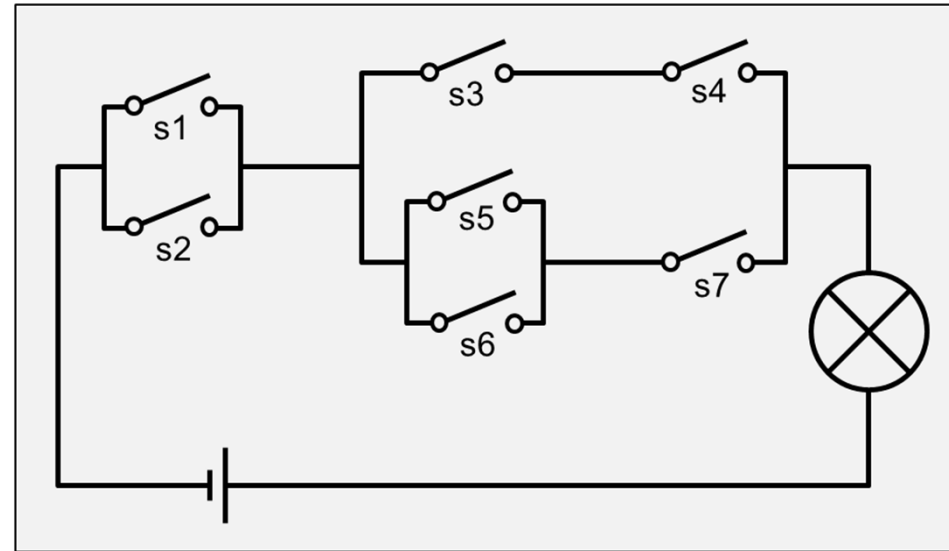


$$z = AB \vee (\overline{A}\overline{B} \vee \overline{A}C)D$$

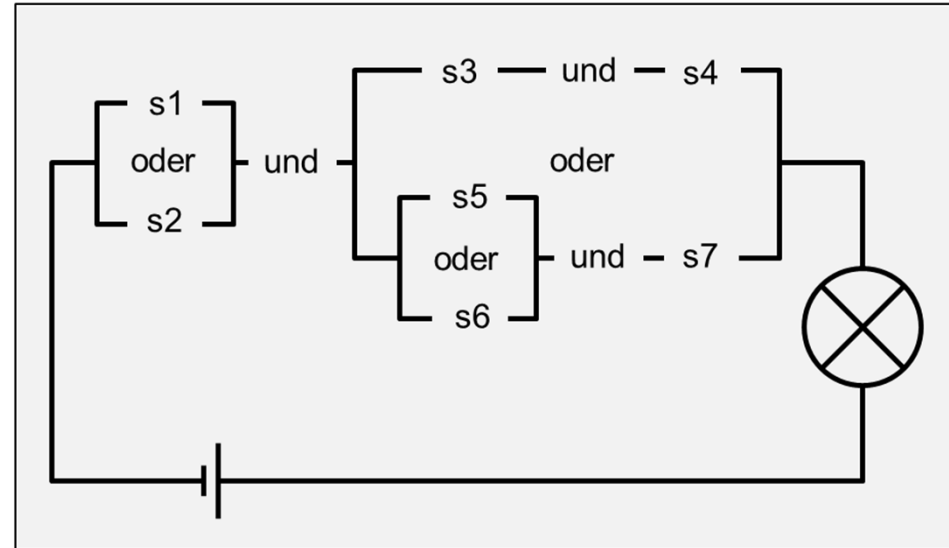
0	0	0	0	:	0
0	0	0	1	:	0
0	0	1	0	:	0
0	0	1	1	:	1
0	1	0	0	:	0
0	1	0	1	:	0
0	1	1	0	:	0
0	1	1	1	:	1
1	0	0	0	:	0
1	0	0	1	:	1
1	0	1	0	:	0
1	0	1	1	:	1
1	1	0	0	:	1
1	1	0	1	:	1
1	1	1	0	:	1
1	1	1	1	:	1

Programmierbeispiel2

Wann leuchtet die Lampe in der Schaltung?



Schalter in Reihe entsprechen einem "und",
parallele Schalter entsprechen einem "oder".




Im C-Code:

```
lampe = (s1 || s2) && ((s3 && s4) || ((s5 || s6) && s7))
```

```
void main()
{
    int s1, s2, s3, s4, s5, s6, s7;
    int lampe;

    printf( "s1 s2 s3 s4 s5 s6 s7\n");

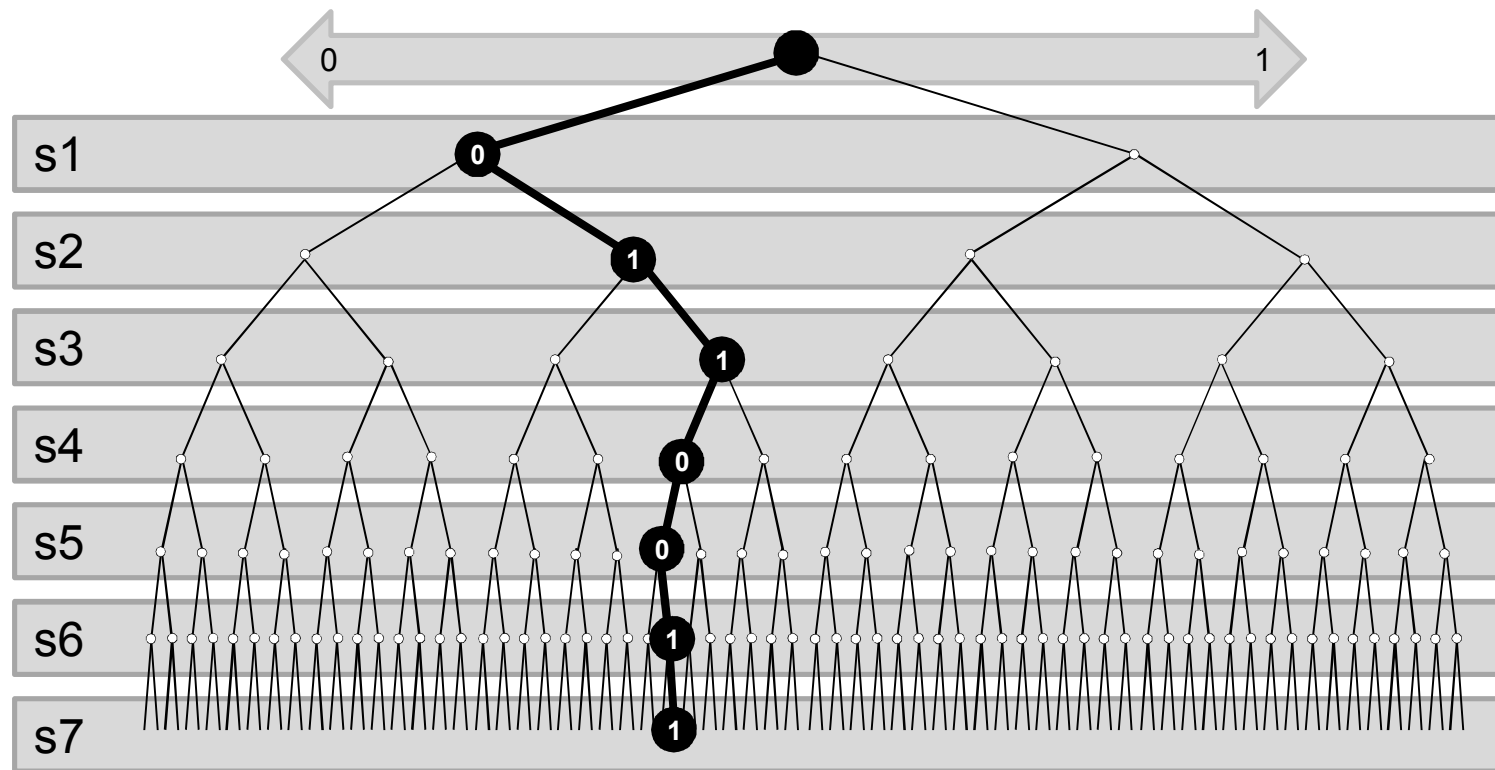
    for( s1 = 0; s1 <= 1; s1 = s1 + 1)
    {
        for( s2 = 0; s2 <= 1; s2 = s2 + 1)
        {
            for( s3 = 0; s3 <= 1; s3 = s3 + 1)
            {
                for( s4 = 0; s4 <= 1; s4 = s4 + 1)
                {
                    for( s5 = 0; s5 <= 1; s5 = s5 + 1)
                    {
                        for( s6 = 0; s6 <= 1; s6 = s6 + 1)
                        {
                            for( s7 = 0; s7 <= 1; s7 = s7 + 1)
                            {
                                lampe = (s1||s2)&&((s3&s4)||((s5||s6)&&s7));
                                if( lampe == 1)
                                    printf( " %d %d %d %d %d %d %d\n", s1, s2, s3, s4, s5, s6, s7);
                            }
                        }
                    }
                }
            }
        }
    }
}
```



s1	s2	s3	s4	s5	s6	s7
0	1	0	0	0	1	1
0	1	0	0	1	0	1
0	1	0	0	1	1	1
0	1	0	1	0	1	1
0	1	0	1	1	0	1
0	1	0	1	1	1	1
0	1	1	0	0	1	1
0	1	1	0	1	0	1
0	1	1	0	1	1	1
0	1	1	1	0	0	0
0	1	1	1	0	0	1
0	1	1	1	0	1	0
0	1	1	1	0	1	1
0	1	1	1	1	0	0
0	1	1	1	1	0	1
0	1	1	1	1	1	1

Kombinatorische Explosion

Mit jedem Schalter verdoppelt sich die Zahl der Schalterkombinationen



Bei n Schaltern ergeben sich 2^n Schalterstellungen. Für $n = 250$ sind das

1809251394333065553493296640760748560207343510400633813116524750123642650624

verschiedene Stellungen. Das ist mehr als die geschätzte Anzahl der Atome im Universum.