Boolesche Algebra B

Constraintsystem

- *Signatur*:
  - Boolesche Variablen: z.B. $X, Y, Z, \ldots$
  - Konstanten: 0 und 1 (Wahrheitswerte)
  - Funktionssymbole: z. B.
    $\neg, \sqcap, \sqcup, \oplus, \rightarrow, \leftrightarrow$
  - Constraintsymbole: $=_B$, *true* und *false*
- *Constrainttheorie*: als Wertetabelle

X	Y	$\neg X$	$X \sqcap Y$	$X \sqcup Y$	$X \oplus Y$	$X \leftrightarrow Y$
0	0	1	0	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	1

- *Erlaubte Constraints*:

$$C ::= \mathit{true} \mid \mathit{false} \mid X =_B Y \mid \neg X =_B Y \mid X \odot Y =_B Z \mid C \wedge C$$

  - X, Y und Z sind Konstanten oder Variablen
  - $\odot$ steht für die zweistelligen Funktionssymbole.

$s =_B t$ ist erfüllt wenn die Booleschen Terme s und t den gleichen Wahrheitswert haben.

Mehrere Ansätze zum Lösen Boolescher Constraints.

Das Entscheidungsproblem, ob ein Boolesches Constraint erfüllt ist (SAT), ist NP-vollständig, d.h. im schlimmsten Fall ist mit exponentieller Komplexität zu rechnen.

Stand 1999

Anmerkung

Boolesche Constraints leiten sich aus der Booleschen Algebra (auch: Schaltalgebra) ab. Die Constraints des Constraintsystems B sind logische Formeln über Variablen, die nur die Werte 1 oder 0 annehmen können, z. B. $(X \sqcup Y =_B 1)$. Diese Booleschen Variablen werden in der Literatur auch als binäre (Schalt-)Variablen bezeichnet.

Konstanten 1 und 0 stehen logisch für die Wahrheitswerte *wahr* und *falsch* bzw. für die Spannungsstufen *H(igh)* und *L(ow)* digitaler Schaltungen.

Die erlaubten Constraints für das Constraintsymbol $\doteq_B$ stellen keine Einschränkung der Allgemeinheit dar, weil jedes Gleichheitsconstraint über beliebigen Booleschen Termen in eine Konjunktion von erlaubten Gleichheitsconstraints transformiert werden kann, indem man Unterterme durch neue Variablen ersetzt und eine entsprechende Gleichheit hinzufügt. Zum Beispiel entspricht das Constraint $A \sqcup (B \sqcap C) \doteq_B B \oplus 0$ dem erlaubten Constraint $(A \sqcup BC \doteq_B B0) \wedge (B \sqcap C \doteq_B BC) \wedge (B \oplus 0 \doteq_B B0)$.

Boolesche Algebra B

Constraint Handling Rules

```
and(X,Y,Z) <=> X=0 | Z=0.
and(X,Y,Z) <=> Y=0 | Z=0.
and(X,Y,Z) <=> X=1 | Y=Z.
and(X,Y,Z) <=> Y=1 | X=Z.
and(X,Y,Z) <=> Z=1 | X=1,Y=1.
and(X,Y,Z) <=> X=Y | Y=Z.
```

Ziel: `and(X,Y,Z), X = 0`

Antwort: `X = 0, Z = 0`

Ziel: `and(X,Y,0), and(X,Z,V), and(Y,V,W)`

Antwort: `and(X,Y,0), and(X,Z,V), and(Y,V,W)`

Anmerkung

In der Implementierung mit CHR verwenden wir für erlaubte Constraints eine relationale Schreibweise, z. B. wird $X \sqcap Y \doteq_B Z$ durch `and(X,Y,Z)` dargestellt:

Die erste Regel besagt, daß das Ziel `and(X,Y,Z)` zu `Z=0` vereinfacht werden kann, wenn bekannt ist, daß `X` den Wert `0` hat. Auch alle anderen Simplifikationsregeln vereinfachen ein Constraint zu einem Gleichheitsconstraint, wenn ein bestimmtes anderes Gleichheitsconstraint folgt.

Die obigen Regeln sind nicht ausreichend, um einen vollständigen Constraintlöser zu erhalten. Zum Beispiel kann auf das Ziel `and(X,Y,0), and(X,Z,V), and(Y,V,W)` keine Regel angewendet werden. Es wird damit nicht erkannt, daß `W=0`. (Wenn `W=1` wäre, müßten alle Variablen den Wert `1` annehmen).

Boolesche Algebra B

Constraintlösen und Suche

```
and(X,Y,Z) <=> X=0 | Z=0.
and(X,Y,Z) <=> Y=0 | Z=0.
and(X,Y,Z) <=> X=1 | Y=Z.
and(X,Y,Z) <=> Y=1 | X=Z.
and(X,Y,Z) <=> Z=1 | X=1,Y=1.
and(X,Y,Z) <=> X=Y | Y=Z.

  enum([]).
  enum([X|L]):- bool(X), enum(L).

    bool(0).
    bool(1).
```

Anmerkung

Wir kombinieren daher das Verfahren mit Enumeration. Ein einfaches Prädikat zur Enumeration für Boolesche Variablen läßt sich wie folgt definieren:

In unserem obigen Beispiel reduziert das Ziel `and(X,Y,0), and(X,Z,V), and(Y,V,W), enum([X,Y,Z,V,W])` zu `and(X,Y,0), and(X,Z,V), and(Y,V,W), bool(X), enum([Y,Z,V,W])`. Die Variable `X` wird an `0` gebunden durch das Ziel `bool(X)` und das Faktum `bool(0)`. Die Booleschen Constraints können zu Gleichheiten vereinfacht werden: `X=0, V=0, W=0, enum([Y,Z,0,0])`. Wenn die Variable `X` durch wiederholtes Rücksetzen an `1` gebunden wird, ergibt sich: `X=1, Y=0, Z=V, W=0, enum([0,Z,Z,0])`.

Boolesche Algebra B

Lösungsverfahren

- Konsistenzverfahren (Lokale Propagierung)
- Theorembeweisen
- Ganzzahlige Programmierung mit 0/1-Variablen
- Gleichungslösen
- Endliche Bereiche

Anmerkung

- Konsistenzverfahren (Lokale Propagierung)
  Wie hier. Die Booleschen Constraints propagieren lokal Gleichheitsconstraints. Um globale Konsistenz zu sichern Enumeration notwendig.
- Theorembeweisen
  Oft werden Varianten der Resolution auf Boolesche Constraints in Klauselnormalform angewendet (z. B. in Prolog III). Problem, alle Lösungen explizit zu generieren.
- Ganzzahlige Programmierung mit 0/1-Variablen
  Dies ist ein Spezialgebiet der mathematischen Programmierung (engl. Operations Research). Oft kommen Varianten der linearen Programmierung zum Einsatz, vor allem in spezialisierten Werkzeugen. Optimierung, aber selten inkrementell, für CLP nicht direkt geeignet.
- Gleichungslösen
  Durch Boolesche Unifikation wird eine allgemeinste Lösung berechnet (in CHIP angewendet). Oft exponentiell groß, mit vielen Hilfsvariablen. Einzellösungen durch Enumeration.
- Endliche Bereiche
  Dabei übersetzt man Boolesche Constraints in Constraints über Endlichen Bereichen, die nur die Werte 0 oder 1 annehmen können. Gleichauf mit lokaler Propagierung, wenn es darum geht, einzelne Lösungen zu berechnen.

Boolesche Algebra B

Projektion

Boolesche Constraints in Konjunktiver Normalform:

$$(\neg X \sqcup Y) \sqcap (\neg X \sqcup Z) \sqcap (X \sqcup \neg Y \sqcup \neg Z)$$

Klauselmengen:

$C^0 = \{\neg X \sqcup Z\}$	Y kommt nicht vor
$C^+ = \{\neg X \sqcup Y\}$	Y kommt positiv vor
$C^- = \{X \sqcup \neg Y \sqcup \neg Z\}$	Y kommt negativ vor

Kombination von C^+ und C^-:

$$\{\neg X \sqcup \neg Z \sqcup X\}$$

Projektion auf X und Z:

$$(\neg X \sqcup \neg Z \sqcup X) \sqcap (\neg X \sqcup Z)$$

Äquivalent zu

$$(\neg X \sqcup Z)$$

Boolesche Algebra B

Projektion

```
project(X), c(L1), c(L2) ==>
          member(+X,L1), member(-X,L2) |
                    select(+X,L1,L3),
                    select(-Y,L2,L4),
                    X==Y,
                    append(L3,L4,L5),
                    c(L5).

project(X) \ c(L) <=> member(+X,L) | true.
project(X) \ c(L) <=> member(-X,L) | true.
project(X) <=> true.
```

Tautologien Löschen:

```
c(L) <=> member(+X,L), member(-X,L) | true.
```

Boolesche Algebra B

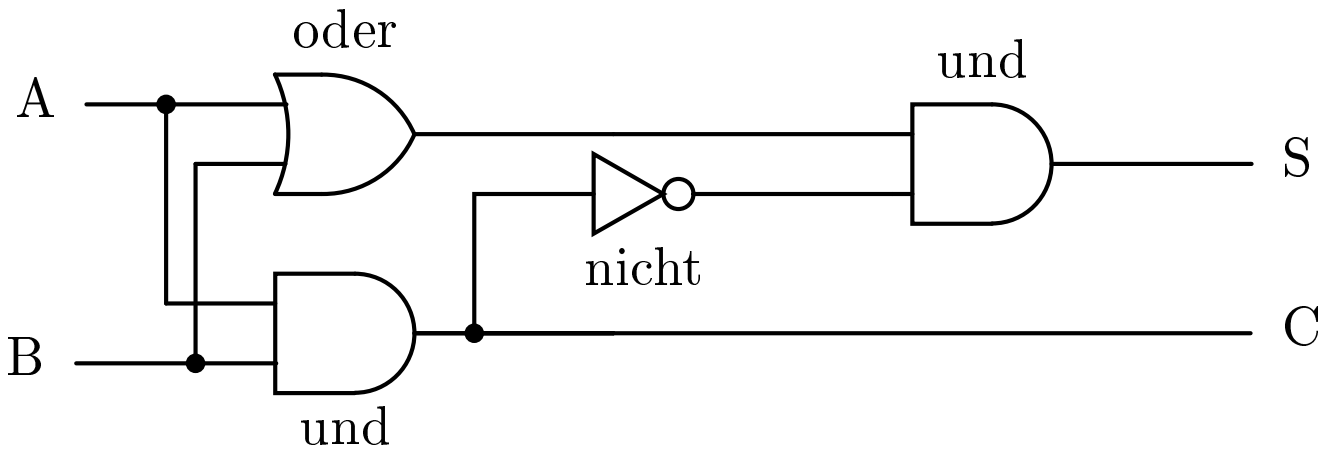
Projektion

```
?- c([-x,+y]), c([-x,+z]), c([+x,-y,-z]), project(y).

c([-(x),+(z)])


?- c([-x,+y]), c([-x,+z]), c([+x,-y,-z]), project(z).

c([-(x),+(y)])


?- c([-x,+y]), c([-x,+z]), c([+x,-y,-z]), project(x),
   project(y).

yes

?- c([-x,+y]), c([+x]), c([-y]), project(x),
   project(y).

c([])
```

Boolesche Algebra B

Anwendung: Modellierung Digitaler Schaltungen

Halb-addierer



Erwartetes Verhalten (Wertetabelle)

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Anmerkung

Im folgenden Beispiel soll ein Halb-Addierer verifiziert werden. Dies ist eine digitale Schaltung, die zwei einstellige Binärzahlen (Bits) addiert. Der Halb-Addierer `add/4` ist durch folgende Wertetabelle spezifiziert und durch folgende Schaltung implementiert. `add(A,B,S,C)` trifft genau dann zu, wenn das Bit `S` die Einerstelle der Summe von `A` und `B` ist und das Bit `C` der Überlauf ist.

Anwendungsbeispiel: Digitale Schaltungen

```
add(A,B,S,C) <=>
    or(A,B,X1),
    and(A,B,C),
    neg(C,X3),
    and(X1,X3,S).
```

Ziel: `add(1,0,S,C)`

Antwort: `S=1,C=0`

Ziel: `add(A,B,S,C), C=1`

Antwort: `A=1, B=1, S=0, C=1`

Ziel: `add(A,B,S,C), C=0`

Antwort: `or(A,B,S), and(A,B,0)`

Anmerkung

Zum Verifizieren der Schaltung repräsentiert man jedes Gatter der Schaltung als Boolesches Constraint:

Man betrachtet alle möglichen Kombinationen von Eingabewerten und kontrolliert das Ergebnis anhand der Spezifikation durch die Wertetabelle. Bei größeren Schaltungen versucht man sich auf möglichst wenige Tests zu beschränken, die dennoch viele Fehler abdecken, da das Durchprobieren aller Eingabewerte wegen des exponentiellen Anwachsens der Kombinationen mit der Zahl der Eingabevariablen nicht möglich ist.

Betrachten wir noch die Eigenschaften von `add/4` als Constraint. Das Ziel `add(A,B,S,C), C=1` reduziert zu `A=1, B=1, S=0, C=1`: Weil `C` den Wert `1` hat, muß die Ausgabe `X3` des Nicht-Gatters `0` sein. Weil `C` auch die Ausgabe von einem Und-Gatter ist, müssen die Eingaben `A` und `B` den Wert `1` haben. Damit läßt sich `X1` berechnen und somit `S`.

Boolesche Algebra B

Anwendung: Fehleranalyse in Digitaler Schaltungen

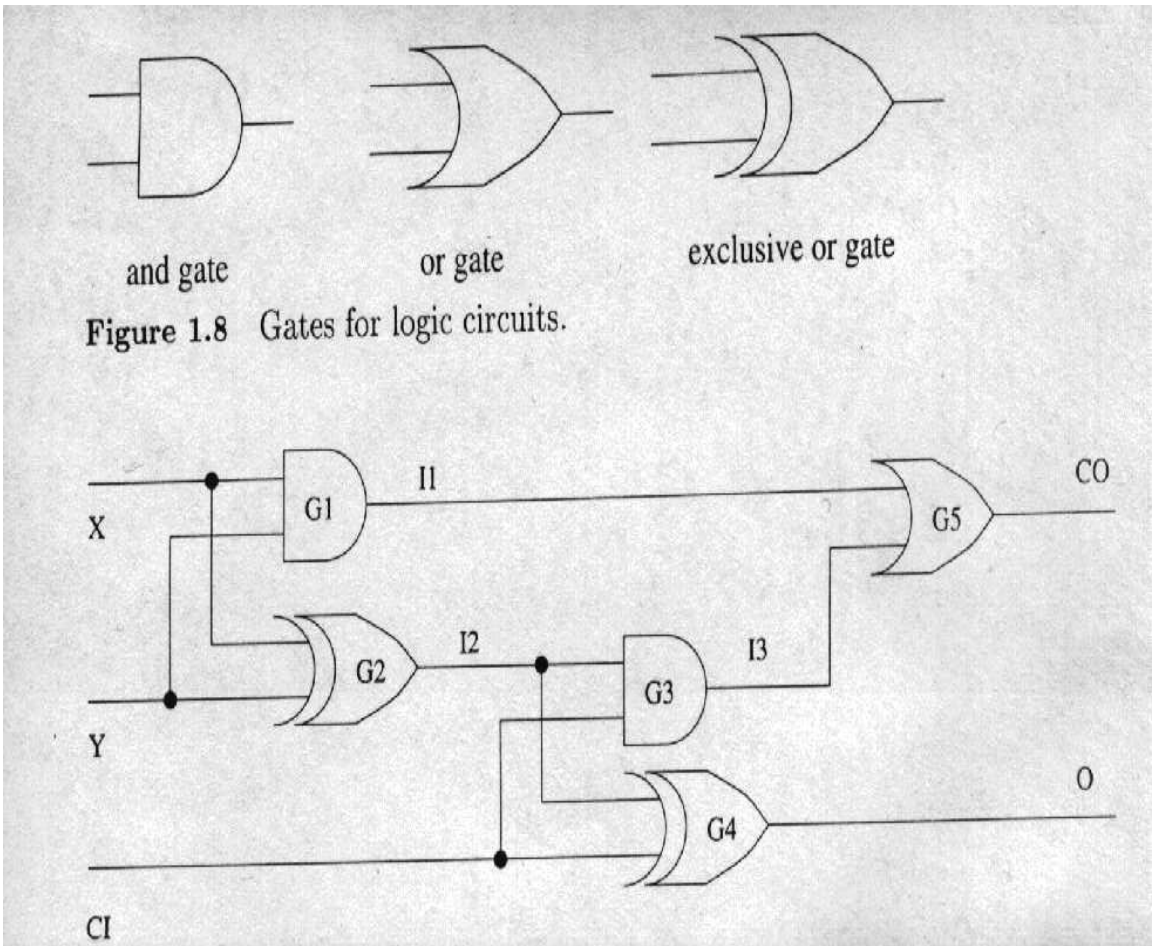


Abbildung 1: Volladdierer

Boolesche Algebra B

Anwendung: Fehleranalyse in Digitaler Schaltungen

Idee: Jedem Gatter G_i eine Variable F_i zuordnen

- Falls $F_i = 1$, dann ist G_i defekt
- Falls $F_i = 0$, dann ist G_i nicht defekt

Verhalten des Addierers: Gatter funktioniert, falls nicht defekt:

$\neg F_1 \rightarrow (I_1 \leftrightarrow X \sqcap Y)$	Und-Gatter G_1
$\neg F_2 \rightarrow (I_2 \leftrightarrow X \oplus Y)$	Xor-Gatter G_2
$\neg F_3 \rightarrow (I_3 \leftrightarrow I2 \sqcap CI)$	Und-Gatter G_3
$\neg F_4 \rightarrow (O \leftrightarrow I_2 \oplus CI)$	Xor-Gatter G_4
$\neg F_5 \rightarrow (CO \leftrightarrow I_1 \sqcup I_3)$	Oder-Gatter G_5

Boolesche Algebra B

Anwendung: Fehleranalyse in Digitaler Schaltungen

Voraussetzung: Maximal ein Gatter ist defekt:

$$\neg(F_1 \sqcap F_2) \wedge \neg(F_1 \sqcap F_3) \wedge \neg(F_1 \sqcap F_4) \wedge \neg(F_1 \sqcap F_5) \wedge$$

$$\neg(F_2 \sqcap F_3) \wedge \neg(F_2 \sqcap F_4) \wedge \neg(F_2 \sqcap F_5) \wedge \neg(F_3 \sqcap F_4) \wedge$$

$$\neg(F_3 \sqcap F_5) \ \wedge \ \neg(F_4 \sqcap F_5)$$

Eingabe:	$X = 0, Y = 0$ und $CI = 1$
Ausgabe:	$O = 0$ und $CO = 1$
Lösung:	$F_1 = 0, F_2 = 1, F_3 = 0, F_4 = 0, F_5 = 0$