4. Übungsblatt zu Logik, Semantik und Verifikation SS 2002

Prof. Dr. Gert Smolka, Dipl.-Inform. Tim Priesnitz
www.ps.uni-sb.de/courses/prog-lsv02/

Abgabe: 6. Mai in der Vorlesungspause

Aufgabe 4.1: Boolesche Algebren (12) Zeigen Sie, dass die folgenden Gleichungen in jeder Booleschen Algebra gelten:

1. $(\overline{x} + y)(x + \overline{y}) = xy + \overline{x}\,\overline{y}$
2. $(x \Leftrightarrow y) = xy + \overline{x}\,\overline{y}$
3. $(\overline{x}y + xz)(\overline{x}u + xv) = \overline{x}yu + xzv$
4. $\overline{x}y + xy = y$
5. $(\overline{x}\,\overline{y} + x\overline{z})\overline{y}\,\overline{z} = \overline{y}\,\overline{z}$
6. $\overline{\overline{x}y + xz} = \overline{x}\,\overline{y} + x\overline{z}$

Hinweis: Zeigen Sie alle Gleichungen von links nach rechts. Benutzen Sie für (5) Gleichung (4). Wenden Sie bei Gleichung (6) nach dem Ausmultiplizieren zuerst Gleichung (5) an und dann das Absorptionsaxiom.

Aufgabe 4.2: Binäre Boolesche Funktionen (9) Diese Aufgabe hilft Ihnen, sich einen Überblick über die Funktionen in $\mathbb{B}^2 \to \mathbb{B}$ zu verschaffen.

(a) (2 Punkte) Realisieren Sie eine Bijektion $(\mathbb{B}^2 \to \mathbb{B}) \to \mathbb{B}^4$ mit zwei Prozeduren

```
to   : (bool * bool -> bool) -> bool * bool * bool * bool
from : bool * bool * bool * bool -> (bool * bool -> bool)
```

Hinweis: $\mathbb{B}^2$ hat 4 Elemente x_1, x_2, x_3, x_4. Eine Funktion $\mathbb{B}^2 \to \mathbb{B}$ kann durch das Tupel $(f\,x_1, \ldots, f\,x_4)$ dargestellt werden.

(b) (1 Punkt) Wieviele Funktionen $\mathbb{B}^2 \to \mathbb{B}$ gibt es?

(c) (6 Punkte) Stellen Sie jede Funktion $f \in \mathbb{B}^2 \to \mathbb{B}$ durch einen möglichst einfachen Booleschen Ausdruck a wie folgt dar:

(i) $f = \lambda\,(x, y) \in \mathbb{B}^2\,.\,a$.

(ii) a ist mit x, y, 0, 1, $\neg$, $\wedge$, $\vee$, $\Rightarrow$ und $\Leftrightarrow$ gebildet.

Aufgabe 4.3: Boolesche Algebren (2) Geben Sie eine Boolesche Algebra $(B, 0, 1, \wedge, \vee, \neg)$ und zwei Elemente $x, y \in B$ an, sodass $x \wedge y = 0$, $x \neq 0$ und $y \neq 0$.

Aufgabe 4.4: Endliche Kompaktheit (6) Wir nehmen an, dass es nur endlich viele Variablen gibt. Eine Menge M von Formeln heißt *unerfüllbar*, wenn es keine Belegung σ mit $\forall A \in M\colon \sigma \in \mathcal{M}(A)$ gibt. Beweisen Sie, dass zu jeder unendlichen unerfüllbaren Menge M von Formeln eine endliche unerfüllbare Teilmenge $M' \subseteq M$ existiert.

Aufgabe 4.5: Teilmengentest (3) Wir stellen Primbäme wie beim letzten Übungsblatt in Standard ML dar. Schreiben Sie eine Prozedur

```
subset: dt * dt -> bool
```

die für zwei Primbäume t_1, t_2 testet, ob $\mathcal{T}(t_1) \subseteq \mathcal{T}(t_2)$. Hinweis: Benutzen Sie die Prozedur `impl` vom letzten Übungsblatt. Lesen Sie im Skript den Abschnitt „Verbandsordnung einer Booleschen Algebra“.

Aufgabe 4.6: Primbäume und Klauseln (18) Wir vereinbaren die folgenden Sprechweisen:

(a) Ein *Literal* ist eine Formel der Form X (Variable) oder $\neg X$ (negierte Variable).

(b) Eine *Klausel* ist eine endliche Menge von Literalen.

(c) Eine *Normalform* ist eine endliche Menge von Klauseln.

(d) Eine Normalform S heißt *konjunktive Normalform (KNF)* für eine Boolesche Menge D, wenn D durch die Formel dargestellt wird, die man erhält, wenn man die Klauseln von S konjunktiv und die Literale jeder Klausel disjunktiv verknüpft.

(e) Eine Normalform S heißt *disjunktive Normalform (DNF)* für eine Boolesche Menge D, wenn D durch die Formel dargestellt wird, die man erhält, wenn man die Klauseln von S disjunktiv und die Literale jeder Klausel konjunktiv verknüpft.

(f) Damit leere Normalformen und Normalformen mit leeren Klauseln wohldefiniert sind, vereinbaren wir, dass eine leere Konjunktion gleichbedeutend mit 1 ist, und eine leere Disjunktion gleichbedeutend mit 0.

Wir stellen Entscheidungsbäume und Normalformen wie folgt in Standard ML dar:

```
type     var     = int  (*  positive numbers only *)
datatype dt      = F | T | D of dt * var * dt
type     literal = int  (*  without zero  *)
type     clause  = int list
type     nf      = clause list
```

Für Variablen verwenden wir nur positive Zahlen, damit wir negierte Variablen durch negative Zahlen darstellen können.

Schreiben Sie die folgenden Prozeduren:

(a) Eine Prozedur `dnf: dt -> nf`, die zu einem Primbaum t eine DNF für die durch t dargestellte Boolesche Menge berechnet.

(b) Eine Prozedur `cnf: dt -> nf`, die zu einem Primbaum t eine KNF für die durch t dargestellte Boolesche Menge berechnet.

(c) Eine Prozedur `witness: dt -> clause`, die zu einem Primbaum t eine Klausel C liefert, so dass die Formel, die man durch Konjunktion der Literale von C erhält, eine nichtleere Teilmenge der von t dargestellten Booleschen Menge ist. Wenn keine solche Klausel existiert, soll die Ausnahme `Match` geliefert werden.

(d) Eine Prozedur `counterwitness: dt -> clause`, die zu einem Primbaum t eine Klausel C liefert, so dass die Formel, die man durch Konjunktion der Literale von C erhält, eine nichtleere Teilmenge des Komplements der von t dargestellten Booleschen Menge ist. Wenn keine solche Klausel existiert, soll die Ausnahme `Match` geliefert werden.

Beachten Sie, dass die Spezifikationen die Prozeduren nicht eindeutig festlegen. Sie haben also Wahlmöglichkeiten. Diese schränken wir dadurch ein, dass wir zusätzlich verlangen, dass alle Klauseln geordnet sind, das heißt, die Variablen sind in Klauseln immer in aufsteigender Reihenfolge angeordnet. Hier sind Beispiele für ein mögliches Verhalten von korrekt implementierten Prozeduren:

```
- val t = D(F, 1, D(T, 2, D(T, 3, F)));
- dnf t;
> val it = [[1, ~2], [1, 2, ~3]] : int list list
- cnf t;
> val it = [[1], [~1, ~2, ~3]] : int list list
- witness t;
> val it = [1, 2, ~3] : int list
- counterwitness t;
> val it = [1, 2, 3] : int list
```

Hinweis: Die Prozeduren vom letzten Übungsblatt werden nicht benötigt und sollen daher nicht verwendet werden.