Aufgabe 2-1

Diese Aufgabe wiederholt noch einmal einfache aussagenlogische und prädikatenlogische Beweise.

Auf der Webseite zur Vorlesung

`http://www.informatik.uni-ulm.de/ki/beweisen-mit-pvs.html`

findet sich die Datei `basic.pvs`. Beweisen Sie eine Auswahl der dort aufgeführten Formeln!

Aufgabe 2-2

In dieser Aufgabe modellieren wir einfache elektronische Bausteine in PVS. Mit einem *Halbaddierer* können zwei einstellige Binärzahlen (Bits) addiert werden. Er besteht aus zwei Eingängen `x` und `y` sowie den beiden Ausgängen `s` und $\texttt{c}_{\texttt{out}}$. Dabei liefert `s` die (rechte Stelle der) Summe der Eingabe-Bits und $\texttt{c}_{\texttt{out}}$ den Übertrag.

a) Definieren Sie in PVS zwei Boolesche Funktionen `HA_s` und `HA_cout` vom Typ `[bool, bool -> bool]` die die betreffenden Ausgabewerte eines Halbaddierers berechnen. (Stellen Sie ggf. zunächst eine Wahrheitstafel auf und lesen Sie daraus die Gleichungen für `s` und $\texttt{c}_{\texttt{out}}$ ab.)

b) Definieren Sie in PVS eine Funktion `bool2nat` die einen (Booleschen) Bit-Wert in die entsprechende natürliche Zahl umrechnet.

c) Formulieren Sie ein Korrektheitstheorem für Ihren Halbaddierer: Ein Halbaddierer ist korrekt, wenn die durch die Ausgänge repräsentierte natürliche Zahl die Summe der (als natürliche Zahl interpretierten) Eingangswerte ist.
Beweisen Sie das Theorem.

Ein Volladdierer (engl. *full adder*) ist eine Erweiterung des Halbaddierers. Ein Volladdierer besitzt zusätzlich zu den beiden Eingängen `x` und `y` einen dritten Eingang $\texttt{c}_{\texttt{in}}$, der den Übertrag aus der vorigen Stelle beinhaltet. Wie beim Halbaddierer liefern die beiden Ausgänge `s` und $\texttt{c}_{\texttt{out}}$ die (rechte Stelle der) Summe der (jetzt drei) Eingabe-Bits bzw. den Übertrag.

d) Modellieren Sie in ähnlicher Weise wie oben einen Volladdierer in PVS und beweisen Sie ein entsprechendes Korrektheitstheorem.

neue PVS-Befehle: `(EXPAND)`, `(EXPAND*)`.

Erweiterungsmöglichkeiten:

e) (Optional) Anstatt die beiden Ausgänge der Addierer einzeln durch jeweils eine Funktion zu modellieren, kann auch eine Funktion definiert werden, die die beiden Ausgabewerte zusammen als ein *Paar* liefert. Modellieren Sie entsprechende Funktionen für den Halb- bzw. den Volladdierer. Beweisen Sie auch die Korrektheit Ihrer Definitionen.

(In PVS kann mit Hilfe der *Projektionen* `p‘1`, `p‘2`, `p‘3`, usw. auf die jeweiligen Komponenten eines Tupels zugegriffen werden.)

f) (Optional) Ein Volladdierer kann auch aus zwei Halbaddierern und einem ODER-Gatter aufgebaut werden. Definieren Sie in dieser Weise eine zweite Version eines Volladdierers und zeigen Sie, dass die beiden Varianten gleich sind.

g) (Optional) Definieren Sie eine Additionsfunktion `add`, die zwei n-stellige Binärzahlen mit Hilfe n Volladdierern addiert.

Beweisen Sie die Korrektheit dieses Addierers für ein festes n, z. B. $n = 4$. (Welches Beweisverfahren würde für beliebige n benötigt werden?)

Aufgabe 2-3

In dieser Aufgabe sollen Mengen in PVS durch ihre charakteristischen Prädikate formalisiert werden. Der folgende Ausschnitt aus der Theorie `Sets` formalisiert die Mengenzugehörigkeit und die Gleichheit auf Mengen:

```
Sets : THEORY
BEGIN
  X   : TYPE+
  SET : TYPE = [X -> bool]            % äquivalent zu: pred[x], set[X] oder setof[X]

  member(x:X,S:SET) : bool = S(x);
  ==(S:SET,T:SET)   : bool = FORALL x: member(x,S) IFF member(x,T)
END Sets
```

Die Menge S wird als ein Prädikat über dem Grundbereich X (dem Typ der Elemente von S) beschrieben. Für ein gegebenes $x \in X$ ist $S(x)$ wahr genau dann, wenn x in der Menge S enthalten ist.

Zwei Mengen S und T sind gleich, wenn sie die gleichen Elemente besitzen, d. h., wenn ein beliebiges x zu S genau dann gehört wenn es auch zu T gehört.

Als Grundgerüst für diese Aufgabe ist die Datei `Sets.pvs` auf der Web-Seite zur Vorlesung hinterlegt. In dieser Datei sind auch noch einige Hinweise zu den Teilaufgaben enthalten, die möglicherweise hilfreich sind.

Hinweis: Sie kommen in den Beweisen mit den Beweisregeln `(SKOSIMP*)`, `(EXPAND)`, `(INST?)`, und `(PROP)` bzw. deren Varianten und Verwandten aus, die in der *Kurzanleitung zu PVS* beschrieben sind.

a) Definieren Sie die leere Menge `emptyset` und die Menge `fullset`, die alle Elemente enthält, in PVS. Zeigen Sie:

```
empty_no_members : LEMMA  NOT member(x, emptyset)
fullset_member   : LEMMA  member(x, fullset)
```

b) Definieren Sie die Mengen `add(x,S)` und `remove(x,S)`, die aus einer Menge `S` dadurch entstehen, dass ein Element `x` der Menge hinzugefügt bzw. aus ihr entfernt wird. Zeigen Sie:

```
member_add        : LEMMA  member(x, add(x, S))
add_idempotent    : LEMMA  add(x, add(x,S)) == add(x,S)
member_remove     : LEMMA  NOT member(x,remove(x,S))
```

c) Definieren Sie ein Prädikat `subset?` für die Teilmengenbeziehung zwischen Mengen. Zeigen Sie:

```
subset_emptyset      : LEMMA  subset?(emptyset, S)
subset_fullset       : LEMMA  subset?(S, fullset)
subset_antisymmetric : LEMMA  subset?(S, T) AND subset?(T, S) IMPLIES S == T
```

d) Definieren Sie Operatoren für Schnitt und Vereinigung von Mengen. Zeigen Sie:

```
union_commutative : LEMMA  union(S, T) == union(T, S)
union_subset      : LEMMA  subset?(S, T) IMPLIES union(S, T) == T
distribute_union_intersection : LEMMA
  union(S, intersection(T, R)) == intersection(union(S, T), union(S, R))
```

e) Definieren Sie Operatoren für das Komplement einer Menge und die Differenz zweier Mengen. Zeigen Sie:

```
complement_emptyset     : LEMMA complement(emptyset) == fullset
demorgan : LEMMA
  complement(intersection(S,T)) == union(complement(S),complement(T))
difference_fullset      : LEMMA difference(S, fullset) == emptyset
difference_intersection : LEMMA difference(S,T) == intersection(S,complement(T))
```

f) (Optional) Beweisen Sie die auch die zahlreichen Lemmata, die am Ende der Theorie `Sets` aufgeführt sind. Können Sie in den Beweisen ein gewisses Muster erkennen? Wie könnte eine komplexe Beweisprozedur aussehen, die diese Art von Formeln automatisch beweisen kann?

In der Dokumentation des PVS Beweisersystems sind im Abschnitt 4.12 ab Seite 78 die mächtigeren Beweisstrategien von PVS beschrieben. Lesen Sie deren Dokumentation und versuchen Sie, mit Hilfe der Ihnen geeignet erscheinenden Strategien manche der Lemmata „schneller“ zu beweisen!

Aufgabe 2-4

In der Vorlesung ist die Definition der Gleichheit wie folgt in der Prädikatenlogik zweiter Stufe angegeben worden:

$$x = y \quad :\Leftrightarrow \quad \forall P : P(x) \Rightarrow P(y)$$

a) Definieren Sie diesen Gleichheitsbegriff in PVS. Verwenden Sie für Ihre Gleichheit in PVS das Symbol `==`.

b) Weisen Sie mit PVS folgende Eigenschaften dieser Gleichheit nach:

- Reflexivität:

  $$\forall x : x \texttt{ == } x$$

- Transitivität:

  $$\forall x, y, z : x \texttt{ == } y \wedge y \texttt{ == } z \Rightarrow x \texttt{ == } z$$

c) Weisen Sie die Substitutivität dieser Gleichheit nach. Unter Substitutivität versteht man, dass Terme an beliebiger Stelle durch gleiche ersetzt werden können. Wir beschränken uns hier auf das Ersetzen innerhalb von Prädikaten:

$$\forall x, y : x \texttt{ == } y \Rightarrow (P(x) \Leftrightarrow P(y))$$

d) Weisen Sie die Symmetrie dieser Gleichheit nach:

$$\forall x, y : x \texttt{ == } y \Rightarrow y \texttt{ == } x$$

Orientieren Sie sich dabei an dem in der Vorlesung vorgestellten Beweis.