GLoIn-Weihnachtsblatt

Zur Vorlesung *Grundlagen der Logik in der Informatik* (WS 2024/25)
Abgabe bis spätestens: 7. Februar 2025

Allgemine Hinweise:

(a) Das Weihnachtsblatt ist ein Bonusblatt und dient dazu, einen Blick über den Tellerrand zu werfen und dabei formales Beweisen zu üben. Man kann ohne Abgabe dieses Blattes 100% der Übungspunkte erreichen.

(b) Für die volle Punktzahl muss der abgegebene Code lesbar sein! Nutzen Sie hierfür Listen (mittels `-`, `*`, `+`) um Induktions- und Äquivalenzbeweise in ihre Fälle zu untergliedern (vgl. Lösung von P6.1 und P10.1).

(c) Abgabe unter Angabe der RRZE-Kennungen (`ab12cdef`) der Team-Mitglieder per E-Mail an:

`cs8-gloin-tutoren@lists.fau.de`

(d) Auf diesem Blatt dürfen Sie zusätzlich die folgenden Coq-Taktiken verwenden:

- `intros x y z.` ist die Kurzform für: `intro x. intro y. intro z.`
- `assumption.` löst das aktuelle Beweisziel, wenn es in den Annahmen vorkommt.

Aufgabe A9.1 Vollständige Induktion in Coq (5 Punkte)

Beweisen Sie Aufgabe A5.1 in Coq, indem Sie in `induction.v` alle fehlenden Beweise einfügen. In der Aufgabe damals hatten wir für $A \in \mathscr{A}$ die Funktion $\phi\colon \mathbb{N} \to \mathscr{F}$ rekursiv definiert:

$$\phi(0) \;=\; A \to B \qquad \phi(k+1) \;=\; \phi(k) \to A$$

Aufgabe: Beweisen Sie in Coq, dass $\phi(k)$ für gerade $k > 0$ herleitbar ist und dass $\phi(k)$ für ungerade k äquivalent zu A ist.

Hinweis: Für die Induktion benötigen Sie lediglich die Taktiken `induction` und `simpl`. Erklärungen dazu finden Sie in den Kommentaren im verlinkten Coq-Quellcode.

Aufgabe A9.2 Die Axiome der Mathematik (7 Punkte)

Die moderne Mathematik ist letztendlich einfach nur eine prädikatenlogische Theorie über der Signatur $\Sigma = \{\in/2\}$ aus einem binären Relationssymbol $\in$ und einer Reihe an Axiomen dazu. Das zentrale Axiom ist das der Extensionalität: zwei Mengen sind genau dann gleich, wenn sie die gleichen Elemente haben:

$$\forall x, y.\, (x = y) \leftrightarrow (\forall e.\, e \in x \leftrightarrow e \in y)$$

Die heute gängigen Axiome wurden von Ernst Zermelo (1871-1953) und Abraham Fraenkel (1891-1965) kurz nach der Jahrhundertwende erarbeitet. Es sind insgesamt 8 Axiome (zwei davon sind Schemata), die beschreiben, auf welche Arten sich neue Mengen bauen lassen. Zum Beispiel:

- Für beliebige Mengen x und y existiert $\{x, y\}$: $\quad \forall x, y.\, \exists z.\, \forall e.\, e \in z \leftrightarrow e = x \vee e = y$
- Jede Menge x hat eine Potenzmenge $\mathcal{P}(x)$: $\quad \forall x.\, \exists z.\, \forall s.\, s \in z \leftrightarrow \forall e. e \in s \to e \in x$
- Für jede Menge x (intuitiv „von Mengen“) existiert die große Vereinigung $\bigcup x$:

$$\forall x.\, \exists z.\, \forall e.\, e \in z \leftrightarrow \exists y.\, e \in y \wedge y \in x$$

- Das Schema der Mengenkomprehension liefert für jede Formel $\phi(x)$ (mit einem Platzhalter x) ein Axiom, das besagt, dass sich jede Menge y auf $\{x \in y \mid \phi(x)\}$ einschränken lässt, also der Menge der Elemente von y die ϕ erfüllen:

$$\forall y. \exists z.\ \forall x.\, x \in z \leftrightarrow x \in y \wedge \phi(x)$$

Da es unhandlich ist, mit den Existenzquantoren zu arbeiten, werden wir für diese Aufgabe die Existenzquantoren durch Funktionssymbole (z.B. $\mathcal{P}$ für Potenzmenge) beschreiben (siehe *Skolemisierung* später in der Vorlesung).

Aufgabe: Beweisen Sie durch Vervollständigung von `gloin_sets_axiomatic.v` in Coq:

(a) Es existiert eine leere Menge. *1 Punkt*

(b) Für beliebige Mengen x und y existiert die Schnittmenge $x \cap y$. *2 Punkte*

(c) Für beliebige Mengen x und y existiert die Vereinigungsmenge $x \cup y$. *2 Punkte*

(d) Die Vereinigung der Potenzmenge einer beliebigen Menge x ist die Menge x selbst: *2 Punkte*

$$\bigcup \mathcal{P}(x) = x.$$

Aufgabe A9.3 Curry-Howard-Korrespondenz (4 Punkte)

Coq und viele andere Theorembeweiser arbeiten intern mit der *Curry-Howard-Korrespondenz*, die besagt, dass Beweisen das gleiche wie (funktionales) Programmieren ist. Wenn Sie in Coq einen Beweis mittels Taktiken aufschreiben, wird intern ein Programm konstruiert, das man genauso gut auch manuell direkt als Beweis aufschreiben kann:

Die Grundidee ist, dass `A -> B` nicht nur als *Implikation* verstanden werden kann, sondern in Coq tatsächlich *Funktionen/Programme die einen Parameter vom Typ* `A` *erwarten und Rückgabewert* `B` *haben*, ähnlich wie die Menge $A \to B$ aller Funktionen von A nach B. Um eine Funktion von `A` nach `B` in Coq zu definieren, schreibt man

```
(fun (a:A) ⇒ ... (* hier etwas vom Typ B *) ... )
```

Beispielsweise können wir wie folgt die Identitätsfunktion definieren:

```
Parameters A B C D : Prop.
Definition def_id : A → A := (fun a ⇒ a). (* aus dem Kontext ist 'a : A' klar. *)
```

Da der Funktionstyp selber nur ein Typ ist, können wir Funktionen auch schachteln:

```
Definition const : A → (B → A) := (fun a ⇒ (fun b ⇒ a)).
```

Diese Funktion beweist die Implikation $A \to (B \to A)$, oder in Coq:

```
Theorem thm_const : A → (B → A).
Proof. exact const. Qed.
```

Wenn andersherum eine Funktion `f: A→ B` und ein `a` vom Typ `A` gegeben ist, dann ist die *Funktionsanwendung* `f a` vom Typ `B`. Konjunktion, Disjunktion und $\bot$ sind *Datentypen*, die die entsprechenden Intro- und Eliminationsregeln als Funktionen mitbringen. Detailierte Erklärungen finden Sie in der in der Coq-Vorlage `curryhoward.v`.

Aufgabe: Beweisen Sie die folgenden Formeln als Funktion (ohne Verwendung von `apply NNPP` oder ähnlichem) und ergänzen Sie sie in `curryhoward.v`:

(a) $(A \vee B \to C) \to (A \to C) \wedge (B \to C)$

(b) $(A \to (B \to C)) \to (A \wedge B \to C)$

(c) $\neg\neg(A \vee \neg A)$

(d) $\neg\neg A \wedge \neg\neg B \to \neg\neg(A \wedge B)$