TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen WS 2016/17
Einführung in die Informatik 2 **Übungsblatt 1**
Prof. Dr. Helmut Seidl, Julian Kranz, Julian Brunner, Nico Hartmann

Aufgabe 1.1 Tutoraufgabe: Umgang mit Quantoren

Beim Umgang mit Quantoren in Beweisen können die folgenden Situationen auftreten:

1. Zu zeigen: $\forall\, x.\ P(x)$
2. Zu zeigen: $\exists\, x.\ P(x)$
3. Als Annahme: $\forall\, x.\ P(x)$
4. Als Annahme: $\exists\, x.\ P(x)$

Dabei steht $P(x)$ für eine beliebige Formel, in der x vorkommen kann. Falls P wieder Quantoren enthält, so können diese anschließend weiter zerlegt werden. So kann die obige Liste mit jeder Kombination und Schachtelung von Quantoren umgehen.

Beschreiben Sie für jede der genannten Situationen, wie im Beweis vorgegangen werden muss. Ist in den Vorgehensweisen eine Symmetrie oder ein Muster zu erkennen?

Lösungsvorschlag 1.1

In den beschriebenen Situationen wird wie folgt vorgegangen:

1. Eine neue Variable a wird eingeführt. Dann wird $P(a)$ bewiesen.
2. Ein beliebiger Term t wird gewählt. Dann wird $P(t)$ bewiesen.
3. Ein beliebiger Term t wird gewählt. Dann wird $P(t)$ angenommen.
4. Eine neue Variable a wird eingeführt. Dann wird $P(a)$ angenommen.

Es fällt auf, dass sich die beiden Quantoren $\forall$ und $\exists$ genau gegensätzlich verhalten. Bei beiden muss im einem Fall eine neue, unbekannte Variable verwendet werden (schwierig), während im anderen Fall ein beliebiger Term gewählt werden kann (einfach). Jedoch tritt bei $\forall$ der erste Fall beim Zeigen und der zweite in der Annahme auf, während es bei $\exists$ genau umgekehrt ist. Intiutiv ist es einfach, etwas *für alle* x annehmen zu können oder nur *für ein* x beweisen zu müssen. Auf der anderen Seite ist es schwierig, etwas *für alle* x beweisen zu müssen oder nur *für ein* x annehmen zu können.

Aufgabe 1.2 Tutoraufgabe: Drinker Paradox

Das Drinker Paradox besagt: In jedem nichtleeren Pub (P) gibt es eine Person, sodass wenn diese Person trinkt (D), dann trinken alle Personen im Pub. Ausgedrückt als logische Formel:

$$\forall\, P.\ (P \neq \{\} \implies \exists\, x \in P.\ (D(x) \implies \forall\, y \in P.\ D(y)))$$

Zeigen Sie die Gültigkeit dieser Formel. Warum ist die Formel wahr, obwohl die im Text formulierte Aussage absurd klingt?

Bemerkungen
$\forall\, x \in A.\ P(x)$ ist eine Abkürzung für $\forall\, x.\ (x \in A \implies P(x))$.
$\exists\, x \in A.\ P(x)$ ist eine Abkürzung für $\exists\, x.\ (x \in A \land P(x))$.

Lösungsvorschlag 1.2

Wir gehen wie in Aufgabe 1.1 besprochen vor und zerlegen die Formel schrittweise in ihre Einzelteile. Die zu beweisende Formel ist allquantifiziert, deswegen führen wir einen neue Variable P ein und beweisen den Rest der Formel. Der Rest der Formel ist eine Implikation, weswegen wir $P \neq \{\}$ (1) annehmen und wieder den Rest der Formel als neues Beweisziel verwenden. Zu beweisen ist also jetzt noch $\exists\, x \in P.\ (D(x) \implies \forall\, y \in P.\ D(y))$. An dieser Stelle machen wir eine Fallunterscheidung nach $\forall\, x \in P.\ D(x)$.

Im ersten Fall können wir also $\forall\, x \in P.\ D(x)$ (2) annehmen. Aus (1) folgern wir $\exists\, x.\ x \in P$ (3). In (3) haben wir nun einen Existenzquantor in einer Annahme und können eine neue Variable a einführen, für die wir $a \in P$ (4) annehmen dürfen. Da wir eine existenzquantifizierte Formel zu beweisen haben, können wir x beliebig instanziieren, in diesem Fall mit a. Es bleibt also zu zeigen: $a \in P \land (D(a) \implies \forall\, y \in P.\ D(y))$. Den ersten Teil der Konjunktion haben wir bereits durch (4). Für den zweiten Teil nehmen wir $D(a)$ (5) an und zeigen $\forall\, y \in P.\ D(y)$. Die Annahme (5) wird in diesem Fall nicht benötigt, da wir das zu Beweisende bereits direkt durch die Annahme der Fallunterscheidung (2) haben.

Im zweiten Fall dürfen wir $\neg\, (\forall\, x \in P.\ D(x))$ (6) annehmen. Mit De Morgan's Regeln und (6) folgern wir $\exists\, x \in P.\ \neg D(x)$ (7). Damit können wir eine neue Variable b einführen, für die wir $b \in P \land \neg D(b)$ (8) annehmen dürfen. Wir teilen (8) auf in $b \in P$ (9) und $\neg D(b)$ (10). Wie im ersten Fall instanziieren wir x im Beweisziel, diesmal mit b. Es bleibt also zu zeigen: $b \in P \land (D(b) \implies \forall\, y \in P.\ D(y))$. Der erste Teil der Konjunktion folgt direkt aus (9). Für den zweiten Teil nehmen wir $D(b)$ (11) an und zeigen $\forall\, y \in P.\ D(y)$. An dieser Stelle haben wir einen Widerspruch zwischen (10) und (11), sodass wir False (12) herleiten können. Da aus False alles folgt, können wir mit Hilfe von (12) das zu Zeigende beweisen.

Aufgabe 1.3 Tutoraufgabe: Assertions mit binärer Suche

Implementieren Sie den Algorithmus zur binären Suche anhand folgender Methodensignatur in Java:

```
public static int binarysearch(int[] a, int key, int imin, int imax)
```

Die Methode erwartet als Parameter ein Integer-Array `a` auf dem nach dem Schlüssel `key` gesucht werden soll. Als Suchgrenzen auf dem Array dienen die Parameter `imin` und `imax`. Der Rückgabewert `n` der Methode ist entweder der Index in dem Array wo gilt `a[n] == key` oder `-1`, falls der Schlüssel nicht in dem Array liegt oder die Länge des Arrays `0` ist.

Schreiben Sie dafür eine naive Implementierung, die nicht die Eingabeparameter auf mögliche Fehler überprüft und fügen Sie danach `assert`-Statements ein um diese Fehler auszuschließen.

Lösungsvorschlag 1.3

```
public static int binarysearch(int[] a, int key, int imin, int imax) {
  assert a != null;

  if (a.length == 0) return -1;

  assert imin >= 0;
```

```
  assert imax >= 0;
  assert imax < a.length;
  assert (new IsSorted(a) != null);

  if (imax < imin) return -1;
  else {
    int imid = imin + ((imax - imin) / 2);
    // im Gegensatz zu (imin+imax)/2 ist das sicher gegen Integer-Overflows

    if (a[imid] > key)
      return binarysearch(a, key, imin, imid - 1);
    else if (a[imid] < key)
      return binarysearch(a, key, imid + 1, imax);
    else
      return imid;
  }
}
```

Aufgabe 1.4 Tutoraufgabe: Installation von Ocaml

In wenigen Wochen wird die Vorlesung sich mit funktionaler Programmierung in der Sprache `Ocaml` beschäftigen. Wir wollen daher die übrige Zeit in dieser Tutorübung dazu nutzen, die nötige Software auf Ihrem Computer zu installieren.

Sie benötigen mindestens den Ocaml-Compiler (`ocamlc`) sowie die interaktive Ocaml-Shell (`ocaml`). Eine Anleitung zur Installation befindet sich unter `https://ocaml.org/docs/install.html` für verschiedene Betriebssysteme.

Etwas mehr Komfort bei der Entwicklung bietet eine IDE, wie z.B. Eclipse. Um Eclipse mit Ocaml zu nutzen, müssen Sie Eclipse zunächst von `http://www.eclipse.org` herunterladen. Klicken Sie anschließend im Menü *Help* auf *Eclipse Marketplace*. Installieren Sie im sich folgend öffnenden Fenster das Plugin *OcaIDE*. Nach einem Neustart von Eclipse steht ihnen der Projekttyp *Ocaml Managed Project* zur Verfügung. Erstellen Sie ein solches Projekt und fügen Sie eine neue Datei *main.ml* mit dem Inhalt

```
print_string "Hello world!\n";;
```

hinzu. Öffnen Sie nun die Projekteigenschaften und wählen Sie den Reiter *Ocaml Project*. Hier können Sie wählen, für *main.ml* ein ausführbares Programm zu erzeugen. Wählen Sie außerdem *native* als *Compilation mode*. Nun sollten Sie in der Lage sein, das Projekt zu bauen und auszuführen.

Bitten Sie Ihren Tutor um Hilfe, falls Sie mit der Installation nicht klarkommen. Sie können auch andere Studenten fragen, die das gleiche Betriebssystem wie Sie nutzen!

Allgemeine Hinweise zur Hausaufgabenabgabe

Die Hausaufgabenabgabe bezüglich dieses Blattes erfolgt ausschließlich über Moodle. Die Abgabefrist finden Sie ebenfalls in Moodle. Bitte vergewissern Sie sich nach der Abgabe selbstständig, dass alle Dateien erfolgreich auf Moodle eingestellt wurden. Die Abgaben der Theorieaufgaben[1] müssen in Form einer einzigen PDF-Datei erfolgen. Nutzen Sie z.B. ImageMagick[2], um aus eingescannten Bildern ein PDF zu erzeugen. Achten Sie bei eingescannten Bildern darauf, dass diese im PDF-Dokument eine korrekte Orientierung haben. **Abgaben, die sich nicht in beschriebenen Formaten befinden, können nicht korrigiert oder bewertet werden!**

Aufgabe [6 Punkte] **Hausaufgabe: Aussagenlogik**
Zeigen Sie:

1. $(A \vee \neg\,(B \wedge A)) \wedge (C \vee D \vee C) \iff C \vee D$
2. $(A \vee B \vee C) \wedge (\neg A \vee C) \iff \neg A \wedge B \vee C$
3. $(A \implies B) \iff (\neg B \implies \neg A)$
4. $(A \wedge B \iff A) \iff (B \iff A \vee B)$

Lösungsvorschlag 1.5

1. Zu zeigen: $(A \vee \neg\,(B \wedge A)) \wedge (C \vee D \vee C) \iff C \vee D$

$$\begin{array}{ll} (A \vee \neg(B \wedge A)) \wedge (C \vee D \vee C) & \iff \\ (A \vee \neg B \vee \neg A) \wedge (C \vee C \vee D) & \iff \\ (A \vee \neg A \vee \neg B) \wedge (C \vee D) & \iff \\ (True \vee \neg B) \wedge (C \vee D) & \iff \\ True \wedge (C \vee D) & \iff \\ C \vee D & \end{array}$$

2. Zu zeigen: $(A \vee B \vee C) \wedge (\neg A \vee C) \iff \neg A \wedge B \vee C$

$$\begin{array}{ll} (A \vee B \vee C) \wedge (\neg A \vee C) & \iff \\ A \wedge \neg A \vee A \wedge C \vee B \wedge \neg A \vee B \wedge C \vee C \wedge \neg A \vee C \wedge C & \iff \\ False \vee A \wedge C \vee B \wedge \neg A \vee B \wedge C \vee C \wedge \neg A \vee C & \iff \\ \neg A \wedge B \vee A \wedge C \vee B \wedge C \vee \neg A \wedge C \vee True \wedge C & \iff \\ \neg A \wedge B \vee (A \vee B \vee \neg A \vee True) \wedge C & \iff \\ \neg A \wedge B \vee True \wedge C & \iff \\ \neg A \wedge B \vee C & \end{array}$$

3. Zu zeigen: $(A \implies B) \iff (\neg B \implies \neg A)$

$$\begin{array}{ll} (A \implies B) \iff \neg A \vee B & \iff \\ B \vee \neg A & \iff \\ \neg\neg B \vee \neg A & \iff \\ (\neg B \implies \neg A) & \end{array}$$

[1] Theorieaufgaben sind Aufgaben, deren Lösung nicht programmiert wird
[2] `http://www.imagemagick.org/script/index.php`

4. Zu zeigen: $(A \wedge B \iff A) \iff (B \iff A \vee B)$
   Annahme: $A \wedge B \iff A$
   Zu zeigen: $B \iff A \vee B$

$$\begin{aligned} & B && \iff \\ & True \wedge B && \iff \\ & (A \vee True) \wedge B && \iff \\ & A \wedge B \vee True \wedge B && \iff \\ & A \wedge B \vee B && \iff \\ & A \vee B \end{aligned}$$

   Annahme: $B \iff A \vee B$
   Zu zeigen: $A \wedge B \iff A$

$$\begin{aligned} & A \wedge B && \iff \\ & A \wedge (A \vee B) && \iff \\ & (A \vee False) \wedge (A \vee B) && \iff \\ & A \vee False \wedge B && \iff \\ & A \vee False && \iff \\ & A \end{aligned}$$

Bemerkungen zu Aufgabe 1.5
Punkteverteilung: 1 + 2 + 1 + 2

Aufgabe [4 Punkte] **Hausaufgabe: Quantoren**
Zeigen Sie:

$$(\exists x.\, \forall y.\, P(x,y)) \implies (\forall y.\, \exists x.\, P(x,y))$$

Gilt die Umkehrung auch?

Lösungsvorschlag 1.6
Wir gehen wie in Aufgabe 1.1 besprochen vor und zerlegen die Formel schrittweise in ihre Einzelteile. Wir nehmen also $\exists x.\ \forall y.\ P(x,y)$ an und zeigen $\forall y.\ \exists x.\ P(x,y)$. Um den Existenzquantor in der Annahme zu behandeln, wählen wir einen neue Variable a und nehmen $\forall y.\ P(a,y)$ an. Um den Allquantor in der Konklusion zu behandeln, wählen wir eine neue Variable b und müssen nun $\exists x.\ P(x,b)$ zeigen. Da wir jetzt einen Allquantor in der Annahme haben, können wir dort für y einen beliebigen Term einsetzen und entscheiden uns, hier b einzusetzen. Damit haben wir $P(a,b)$. In der Konklusion steht ein Existenzquantor, so das wir dort für x einen beliebigen Term einsetzen dürfen und entscheiden uns, hier a einzusetzen. Damit müssen wir $P(a,b)$ zeigen, was wir bereits getan haben, sodass der Beweis an dieser Stelle abgeschlossen ist.

Die Umkehrung der Implikation gilt nicht, ein Gegenbeispiel ist $P(x,y) = (x = y)$. Es gilt dann $\forall y.\ \exists x.\ x = y$, aber nicht $\exists x.\ \forall y.\ x = y$. Eine Voraussetzung ist natürlich, dass das Universum mehr als einen Wert enthält, da ansonsten $\forall x\, y.\ x = y$ gilt.

Aufgabe [10 Punkte] **Hausaufgabe: Assertions**
Fügen Sie in den Java-Dateien im Ordner `src/assertions/` an den mit `// TODO` markierten Stellen sinnvolle `assert`-Statements ein. Der Sinn der Klassen dieser Aufgabe ist es, bestimmte Eigenschaften im Typ widerzuspiegeln und damit z.B. Anforderungen an die Parameter einer Funktion erzwingen zu können.

Lösungsvorschlag 1.7
Die Lösung befindet sich im Ordner `src/assertions-solution/`.

Bemerkungen zu Aufgabe 1.7
Punkteverteilung:

- `NotNull.java`: 1 Punkt
- `NonEmpty.java`: 1 Punkt
- `IsSorted.java`: 2 Punkte
- `FunConstraints.java`:
  - `addPrecondition`: 2 Punkte
  - `addPostcondition`: 2 Punkte
  - `addConditions`: 2 Punkt