

Praktikum Spezifikation und Verifikation

Sortieren auf Listen

Der erste Teil des Blattes beschäftigt sich mit einem einfachen Sortieralgorithmus auf Listen von natürlichen Zahlen. Ihre Aufgabe ist es, den Sortieralgorithmus in Isabelle zu programmieren und zu zeigen, dass Ihre Implementierung korrekt ist.

Der Algorithmus lässt sich in folgende Funktionen zerlegen:

```
consts
  insert :: "nat ⇒ nat list ⇒ nat list"
  sort   :: "nat list ⇒ nat list"
  le     :: "nat ⇒ nat list ⇒ bool"
  sorted :: "nat list ⇒ bool"
```

Hierbei soll *insert* x *xs* eine Zahl x in eine sortierte Liste *xs* einfügen, und die Funktion *sort* *ys* (aufbauend auf *insert*) die sortierte Version von *ys* liefern.

Um zu zeigen, dass die resultierende Liste tatsächlich sortiert ist, benötigen sie das Prädikat *sorted* *xs*, das überprüft ob jedes Element der Liste kleiner ist als alle folgenden Elemente der Liste. Die Funktion *le* n *xs* soll genau dann wahr sein, wenn n kleiner oder gleich allen Elementen von *xs* ist.

Zeigen Sie zu Beginn folgendes Monotonie-Lemma über *le*. Die Formulierung des Lemmas ist aus technischen Gründen (über die sie später im Praktikum mehr erfahren werden) etwas ungewohnt:

```
lemma [simp]: "x ≤ y ⇒ le y xs → le x xs"
  :
```

Zeigen Sie dann folgendes Korrektheits-Theorem:

```
theorem "sorted (sort xs)"
  :
```

Das Theorem sagt zwar aus, dass Ihr Algorithmus eine sortierte Liste zurückgibt, es gibt Ihnen aber nicht die Sicherheit, dass die sortierte Liste auch die gleichen Elemente wie das

Original enthält. Formulieren Sie deswegen eine Funktion `count xs x`, die zählt, wie oft die Zahl `x` in `xs` vorkommt.

Zeigen Sie:

```
theorem "count (sort xs) x = count xs x"
  :
```

Sortieren mit Bäumen

Der zweite Teil des Blattes beschäftigt sich mit einem Sortieralgorithmus, der Bäume verwendet. Definieren Sie dazu zuerst den Datentyp `bintree` der Binärbäume. Für diese Aufgabe sind Binärbäume entweder leer, oder bestehen aus einem Knoten mit einer natürlichen Zahl und zwei Unterbäumen.

Schreiben Sie dann eine Funktion `tsorted`, die feststellt, ob ein Binärbaum geordnet ist. Sie werden dazu zwei Hilfsfunktionen `tge` und `tle` benötigen, die überprüfen ob eine Zahl grössergleich/kleiner als alle Elemente eines Baumes sind.

Formulieren Sie schliesslich eine Funktion `tree_of`, die zu einer (unsortierten) Liste den geordneten Binärbaum zurückgibt. Stützen Sie sich dabei auf eine Funktion `ins n b`, die eine Zahl `n` in einen geordneten Binärbaum `b` einfügt.

Zeigen Sie:

```
theorem [simp]: "tsorted (tree_of xs)"
  :
```

Auch bei diesem Algorithmus müssen wir noch zeigen, dass keine Elemente beim Sortieren verloren gehen (oder erfunden werden). Formulieren Sie analog zu den Listen eine Funktion `tcount x b`, die zählt, wie oft die Zahl `x` im Baum `b` vorkommt.

Zeigen Sie:

```
theorem "tcount (tree_of xs) x = count xs x"
  :
```

Die eigentliche Aufgabe war es, Listen zu sortieren. Bis jetzt haben wir lediglich einen Algorithmus, der Listen in geordnete Bäume transformiert. Definieren Sie deswegen eine Funktion `list_of`, die zu einen (geordneten) Baum eine (geordnete) Liste liefert.

Zeigen Sie: (**Achtung Update!**)

```
theorem "sorted (list_of (tree_of xs))"
  :
```

```
theorem "count (list_of (tree_of xs)) n = count xs n"
  :
```

Hinweise:

- Bevor Sie mit den Korrektheitsbeweisen anfangen, sollten Sie an einigen konkreten, kleinen Beispielen überprüfen, ob ihre Funktionen das tun, was sie erwarten.
- Die *auto*-Methode kann zwar mit Implikationen umgehen, für dieses Blatt eignen sich aber Gleichungen am besten. Versuchen Sie, alle Lemmas als Gleichungen zu formulieren, und sie so aufzuschreiben (ggf. statt $a = b$ einfach $b = a$), dass die rechte Seite ‘einfacher’ ist als die linke.
- Für den letzten Teil müssen Sie *sorted* auf Listen und *tsorted* auf Bäumen zueinander in Beziehung setzen. Sie benötigen dazu eine Funktion *ge* auf Listen (analog zu *tge* auf Bäumen) und folgendes Lemma (das sie noch beweisen müssen):
$$\text{sorted } (a @ x \# b) = (\text{sorted } a \wedge \text{sorted } b \wedge \text{ge } x \ a \wedge \text{le } x \ b)$$
- Nutzen Sie frühzeitig die Betreuungsangebote, wenn Sie Schwierigkeiten haben. Die Aufgabe lässt sich nicht an einem Nachmittag lösen.
- Wir möchten Schwierigkeit und Aufwand dieser Aufgabe genauer evaluieren. Wir sind v.a. an Fehlern (Zahl und Art) in Funktionsdefinitionen interessiert, auf die Sie während der Beweisarbeit stossen. Heben Sie deswegen bitte falsche Versionen Ihrer Definitionen auf, wenn Sie eine Änderung vornehmen – am einfachsten ist es, die alte Version auszukommentieren, und die neue Version darunter zu schreiben. Kommentare sind in Isabelle in (* und *) eingeschlossen. Die Fehler fließen selbstverständlich *nicht* in Ihre Bewertung mit ein – im Gegenteil, Sie machen es sich und nachfolgenden Generationen einfacher, wenn sie alle Änderungen möglichst genau dokumentieren.

- ▷ **Abgabe Sortieren auf Listen + alle Funktionsdefinitionen: 2. Mai 2002**
- ▷ **Abgabe Rest: 8. Mai 2002**