Karlsruhe Institute of Technology

Lehrstuhl für Programmierparadigmen
Daniel Wasserrab
daniel.wasserrab@kit.edu

Theorembeweiserpraktikum – SS 2010

http://pp.info.uni-karlsruhe.de/lehre/SS2010/tba

Blatt 4: Allgemeine Rekursion Besprechung: 04.05.2010

1 Listen zusammenfügen

In dieser Aufgabe soll eine Funktion *zip* definiert werden, welche zwei Listen durch Verschachtelung zusammenfügt. Beispiel: `zip [a1, a2, a3] [b1, b2, b3] = [a1, b1, a2, b2, a3, b3]` und `zip [a1] [b1, b2, b3] = [a1, b1, b2, b3]`. Verwenden Sie dazu totale Rekursion, also *fun*.

Zeigen Sie, dass *zip* distributiv über *append* (also `@`) ist. Brauchen Sie dazu zusätzliche Prämissen?

2 Merge Sort

Wir arbeiten im Folgenden nur auf Listen über natürlichen Zahlen.

Definieren Sie ein Prädikat *sorted*, welches prüft, ob jedes Element kleiner oder gleich den folgenden ist; `le n xs` ist `True` g.d.w. `n` kleiner oder gleich allen Elementen in `xs`.

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

Implementieren Sie nun *Merge Sort*: Eine Liste wird durch Aufteilung in zwei Listen sortiert, welche einzeln sortiert und wieder zusammengefügt werden.

Definieren Sie mittels *fun* zwei Funktionen

```
consts merge :: "nat list ⇒ nat list ⇒ nat list"
       msort :: "nat list ⇒ nat list"
```

und zeigen Sie

```
theorem "sorted (msort xs)"
oops
```

Sie werden dafür Hilfslemmas über `le` und *sorted* beweisen müssen.

Hinweise:

- Um eine Liste in zwei fast gleichlange Hälften zu zerteilen, können Sie die Funktionen `n div 2`, *take* und *drop* verwenden, wobei `take n xs` die ersten `n` Elemente von `xs` zurückgibt, `drop n xs` den Rest.

- Versuchen Sie erstmal, das Lemma alleine zu lösen und selbst herauszufinden, welche Hilfslemmas Sie dafür brauchen. Falls Sie so nicht weiterkommen, hier ein paar Überlegungen:
  - was muss gelten, damit `merge sorted` ist?
  - was muss gelten, wenn der zweite Parameter von `le merge` ist?
  - wie verhält sich `le`, wenn der erste Parameter kleiner wird?

3 Vollständige Binärbäume

In dieser Übung arbeiten wir mit Skeletten von Binärbäumen, in denen weder die Blätter ("tip") noch die inneren Knoten Daten enthalten:

datatype `tree = Tp | Nd tree tree`

Definieren Sie eine Funktion `tips`, welche die Blätter eines Baumes zählt und eine Funktion `height`, welche die Höhe eines Baumes berechnet (Hinweis: Wenn man die Höhe der Wurzel als 0 definiert, sind die folgenden Beweise einfacher).

Vollständige Binärbäume lassen sich durch folgende Funktion generieren:

```
primrec cbt :: "nat ⇒ tree"
  where "cbt 0       = Tp"
  | "cbt (Suc n) = Nd (cbt n) (cbt n)"
```

Im Folgenden werden wir uns auf diese vollständige Binärbäume konzentrieren.

Anstatt vollständige Binärbäume zu generieren, kann man auch *testen*, ob ein Binärbaum vollständig ist. Definieren Sie eine Funktion `iscbt f` (wobei `f` eine Funktion über `tree` ist), welche `tree`s auf Vollständigkeit überprüft: `Tp` ist vollständig, und `Nd l r` ist vollständig genau dann, wenn `l` und `r` vollständig sind und `f l = f r` gilt.

Wir besitzen nun 3 Funktionen über `tree`s, nämlich `tips`, `height` und `size`. Letztere ist automatisch für jeden Datentyp definiert: Alle Konstruktoren, in denen der Typ nicht rekursiv vorkommt, haben `size` 1, ansonsten werden die `size`s der rekursiven Aufrufe zusammengezählt und um 1 erhöht (z.B. `size Tp = 0`, `size (Nd l r) = size l + size r + 1`). Wir besitzen nun also 3 Ausdrucksweisen der Vollständigkeit: vollständig bzgl. `tips`, vollständig bzgl. `height` und vollständig bzgl. `size`. Zeigen Sie,

- dass diese drei Begriffe dasselbe beschreiben (z.B. `iscbt tips t = iscbt size t`) und
- dass diese drei Begriffe genau diese `tree`s beschreiben, welche von `cbt` generiert werden: das Ergebnis von `cbt` ist vollständig (im Sinne von `iscbt` bzgl. jeder beliebigen Funktion auf `tree`s), und falls ein `tree` vollständig ist im Sinne von `iscbt`, ist er das Resultat von `cbt` (angewandt auf eine bestimmte Funktion – welche?).

Hinweise:

- Überlegen Sie sich und beweisen sie geeignete Beziehungen zwischen `tips`, `height` und `size`. Gelten die Aussagen allgemein oder nur für vollständige Bäume?
- Sie müssen nicht zeigen, dass jeder Begriff gleich jedem anderen ist. Es genügt zu zeigen, dass $A = B$ und $B = C$, daraus folgt $A = C$ trivialerweise. Jedoch besteht die Schwierigkeit darin, herauszufinden, welche dieser Gleichheiten am leichtesten zu beweisen sind.