Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph. D.

SS 2000
9. 5. 2000
Aufgabenblatt 3

Praktikum Spezifikation und Verifikation

1 Tries

1.1 Eine Funktion zum Einfügen und Löschen

In Kapitel 3.4.4 des Tutorials wird eine Fallstudie über sogenannte Tries durchgeführt, das sind Datenstrukturen zur schnellen Indizierung mit Strings, d.h. für gewisse Strings wird in einem zeichenweise verzweigten Baum ein Wert zugeordnet.

Lesen Sie den Abschnitt 3.4.4 im Tutorial.

Der Datentyp der Tries über dem Alphabet-Typ `'a` und dem Werte-Typ `'v` wird dort auf die folgende Weise definiert.

```
datatype ('a, 'v) trie = Trie  "'v option"  "('a * ('a,'v) trie) list"
```

Ein Trie besteht also aus einem optionalen Wert und einer Assoziationsliste, die Zeichen des Alphabets auf Unterbäume abbildet. Zur Modellierung optionaler Werte wird der Datentyp `'a option` verwendet (siehe Tutorial, 2.5.3).

Ferner werden dort zugehörige Selektor-Funktionen `value` und `alist` definiert.

```
consts value :: "('a, 'v) trie ⇒ 'v option"
primrec "value (Trie ov al) = ov"

consts alist :: "('a, 'v) trie ⇒ ('a * ('a,'v) trie) list"
primrec "alist (Trie ov al) = al"
```

Außerdem ist die Funktion `lookup` für Tries gegeben, die mit Hilfe der generischen Such-Funktion `assoc` für Assoziationslisten definiert wird.

```
consts assoc ::  "('key * 'val)list ⇒ 'key ⇒ 'val option"
primrec "assoc [] x = None"
        "assoc (p#ps) x =
```

```
(let (a, b) = p in if a = x then Some b else assoc ps x)"
```

```
consts lookup :: "('a, 'v) trie ⇒ 'a list ⇒ 'v option"
primrec "lookup t [] = value t"
        "lookup t (a#as) = (case assoc (alist t) a of
                              None ⇒ None
                            | Some at ⇒ lookup at as)"
```

Die Funktion `update` fügt einen neuen Wert zu einem String in einen Trie ein, indem sie den alten Wert, der dem String zugeordnet war, überschreibt.

Zunächst nochmal die Definition der Funktion `update` aus dem Tutorial.

```
consts update :: "('a, 'v) trie ⇒ 'a list ⇒ 'v ⇒ ('a, 'v) trie"
primrec
  "update t []     v = Trie (Some v) (alist t)"
  "update t (a#as) v =
     (let tt = (case assoc (alist t) a of
                 None ⇒ Trie None []
               | Some at ⇒ at)
      in Trie (value t) ((a, update tt as v) # alist t))"
```

Das folgende Theorem besagt, dass sich die Funktion `update` so verhält, wie man es erwartet:

```
theorem "∀t v bs. lookup (update t as v) bs =
                  (if as = bs then Some v else lookup t bs)" ⟨proof⟩
```

Definieren Sie eine Funktion

```
consts modify :: "('a, 'v) trie ⇒ 'a list ⇒ 'v option ⇒ ('a, 'v) trie"
```

mit der man sowohl Elemente einfügen als auch löschen kann. Beweisen Sie für diese Funktion eine abgewandelte Version des obigen Theorems.

1.2 Eine optimierte Version der *modify* Funktion: Überschreiben statt Anfügen

Die obige Definition der Funktion `update` hat den Nachteil, dass in der Assoziationliste unnötige Information enthalten sein kann: Wenn man an die Liste ein neues Assoziationspaar zu einem Zeichen anfügt, das schon in der Liste enthalten ist, bleibt das ursprüngliche Paar in der Liste enthalten und ist nur nicht mehr durch die Funktion `assoc` zugänglich.

Diesen Nachteil kann man beheben, indem man eine (noch zu definierende) Funktion

consts *overwrite :: "'a ⇒ 'b ⇒ ('a * 'b) list ⇒ ('a * 'b) list"*

verwendet, die eine Assoziation zu einem Zeichen, das schon in der Liste vorkommt, durch ein neues Paar ersetzt.

Definieren Sie die Funktion *overwrite*, verändern Sie die Funktion *modify*, so dass sie die Funktion *overwrite* verwendet und zeigen Sie, wieder den Zusammenhang zwischen *modify* und *lookup*.

1.3 Eine andere Datenstruktur für Tries: Verwendung partieller Funktionen

Anstelle von Assoziationslisten können zur Modellierung der Verzweigung in einem Knoten auch partielle Funktioen verwendet werden, die die Zeichen des Alphabets auf zugehörige Unterbäume abbilden. Partielle Funktionen kann man mit Hilfe des Typs *'a option* simulieren: Ist *f* eine Funktion vom Typ *'a ⇒ 'b option*, dann ist *f a = Some b*, falls *a* ein Wert *b* zugeodnet werden soll, ansonsten ist *f a = None*.

datatype *('a, 'v) trie = Trie "'v option" "'a ⇒ ('a,'v) trie option"*

Geben Sie entsprechende Selektoren für die veränderte Datenstruktur an, passen Sie die Funktionen *lookup* und *modify* an und beweisen wieder ihren Zusammenhang.

▷ **Abgabe: 16. 5. 2000**