

Übungen zu „Übersetzerbau“ Blatt 9

Aufgabe 33. Implementiere binäre Suchbäume in Tiger.

```
type key = string
type val = int
type tree = { key : key, val : val, left : tree, right : tree }
```

1. Definiere eine Funktion, die den mit einem Schlüssel assoziierten Wert bestimmt.

```
function lookup (t : tree, k : key) : val
```

2. Definiere eine Funktion, die eine Bindung in einen binären Suchbaum einträgt.

```
function insert (t : tree, k : key, v : val)
```

3. Die Prozedur *insert* aus Teil 2 modifiziert notwendigerweise den binären Suchbaum. (Warum?) Programmiere eine *persistente* Variante, die einen neuen Baum konstruiert.

```
function insert (t : tree, k : key, v : val) : tree
```

Aufgabe 34. Die Semantik von Tiger ist nicht formal definiert. Welche offenen Punkte gibt es? Wie würdest du die Semantik jeweils festlegen? Betrachte zum Beispiel

```
let var a := 5
function foo (x : int, y : int) =
  (x := x + 1; x + y)
in foo ((print ("hello world"; a), (a := a + 1; 7)));
a
end
```

Aufgabe 35. Programmiere eine Monade *Action*, die sowohl Zustände als auch Ausnahmen unterstützt.

```
get  :: Action State
set  :: State → Action ()
raise :: Error → Action α
catch :: Action α → (Error → Action α) → Action α
```

Gibt es mehrere Möglichkeiten, die beiden Effekte miteinander zu kombinieren?

Aufgabe 36. Nicht alle Tiger Funktionen benötigen einen statischen Verweis (‘tiger book’, Aufgabe 6.5).

1. Charakterisiere die Funktionen, die ohne einen statischen Verweis auskommen.
2. Gib einen Algorithmus an, der diese Eigenschaft überprüft.