

Übungen zu „Übersetzerbau“ Blatt 7

Aufgabe 25. Der folgende Baumtyp repräsentiert um lokale Definitionen erweiterte arithmetische Ausdrücke.

```
data Expr α = Var α           -- x
             | Num Int        -- n
             | Bin (Expr α) Op (Expr α) -- e1 ⊗ e2
             | Let α (Expr α) (Expr α) -- let x = e in e'
data Op = Add | Sub | Mul | Div | Exp
```

Programmiere folgende Funktionen unter Verwendung des Schemas der strukturellen Rekursion:

- Einen Auswerter:

$$eval :: Expr\ \alpha \rightarrow (\alpha \rightarrow Int) \rightarrow Int$$

- Eine Funktion, die die in einem Ausdruck enthaltenen freien Variablen bestimmt:

$$free :: Expr\ \alpha \rightarrow \{\alpha\}$$

- Eine Funktion, die überprüft, ob ein Ausdruck geschlossen ist (keine freien Variablen enthält):

$$closed :: Expr\ \alpha \rightarrow Bool$$

Aufgabe 26. Programmiere einen „pretty printer“ für den Typ $Expr\ \alpha$ aus Aufgabe 25. Verwende die konkrete Syntax, die in der Typdeklaration angegeben ist. Versuche möglichst wenige Klammern auszugeben, indem du die Bindungsstärken der Operatoren berücksichtigst und die Vereinbarung, dass sich ein **let** so weit nach rechts erstreckt wie möglich (**let** $x = 1$ **in** $x + 2$ entspricht **let** $x = 1$ **in** $(x + 2)$ und *nicht* **let** $x = 1$ **in** $x + 2$).

Aufgabe 27. Programmiere einen Top-down Parser für arithmetische Ausdrücke. Verwende die in der Vorlesung vorgestellten Parser-Kombinatoren. *Hinweis:* verwende für die lexikalische Analyse einen einfachen, von Hand programmierten Scanner.

Aufgabe 28. Programmiere einen Parser für erweiterte reguläre Ausdrücke (siehe auch Aufgabe 9 und 14). Verwende zu diesem Zweck einen Parser-Generator deiner Wahl. Verwende eine möglichst natürliche (mehrdeutige) Grammatik und löse Konflikte durch die Angabe von Prioritäten.