**Lehrstuhl für Programmiersprachen und Übersetzerkonstruktion**
CAU Kiel, Informatik und Prakt. Mathematik
Prof. Dr. M. Hanus, Dr. F. Huch

24118 Kiel
Christan-Albrechts-Platz 4
Tel.: 0431 / 880-7277

# 3. Übung „Übersetzerbau“

## Abgabe am 9. Mai in der Vorlesung

---

**Aufgabe 9** 1+6 Punkte

Wir erweitern die Programmiersprache Simple um folgende Konstrukte:

| | | | |
|---|---|---|---|
| *Stm* | $\longrightarrow$ | `if` *BExp* `then` *Stm* `else` *Stm* | (IfStm) |
| *Stm* | $\longrightarrow$ | `while` *BExp* `do` *Stm* | (WhileStm) |
| *Stm* | $\longrightarrow$ | `nop` | (NopStm) |
| *BExp* | $\longrightarrow$ | *Exp BinCmp Exp* | (CmpBExp) |
| *BExp* | $\longrightarrow$ | `tt` | (TTBExp) |
| *BExp* | $\longrightarrow$ | `ff` | (FFBExp) |
| *BExp* | $\longrightarrow$ | *BExp BinBOp BExp* | (BooleExp) |
| *BinCmp* | $\longrightarrow$ | `=` | (Equal) |
| *BinCmp* | $\longrightarrow$ | `/=` | (NotEqual) |
| *BinCmp* | $\longrightarrow$ | `<=` | (LessEqual) |
| *BinCmp* | $\longrightarrow$ | `>=` | (GreaterEqual) |
| *BinBOp* | $\longrightarrow$ | `&&` | (And) |
| *BinBOp* | $\longrightarrow$ | `\|\|` | (Or) |

a) Erweitern Sie die zugehörigen algebraischen Haskell-Datenstrukturen.

b) Erweitern Sie den Interpreter um die neuen Konstrukte. Für die Interpretation der `while`-Schleife reicht ein einfacher rekursiver Abstieg in der Baumstruktur nicht aus. Überlegen Sie, wie Sie die Schleife im Interpreter abwickeln können, so dass Sie die Schleifen iterieren.

**Aufgabe 10** 3 Punkte

Für reguläre Ausdrücke sind die folgenden Abkürzungen üblich:

- $[a_1, a_2, \ldots, a_n] := (a_1 \mid a_2 \mid \cdots \mid a_n)$ für $a_1, a_2, \ldots, a_n \in \Sigma$.
- $\alpha^+ := \alpha\alpha^*$ für $\alpha \in \mathrm{RA}(\Sigma)$.
- $\alpha? := (\alpha \mid \varepsilon)$ für $\alpha \in \mathrm{RA}(\Sigma)$.

Erweitern Sie das Verfahren nach Thompson (regulärer Ausdruck $\leadsto$ NFA) derart, dass für diese Abkürzungen kleinere NFAs erzeugt werden. Wenden Sie Ihre Erweiterungen auf $\beta = [\mathtt{a}, \mathtt{b}, \mathtt{c}]^+\mathtt{d}? \in \mathrm{RA}(\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\})$ an.

**Aufgabe 11** 2 Punkte

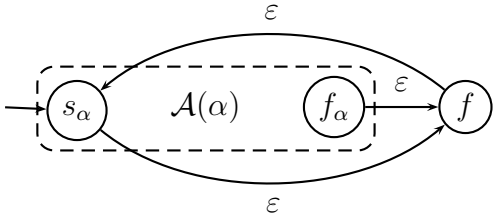
Geben Sie eine reguläre Definition an, die die Gleitpunktkonstanten der Sprache `C` beschreibt:

> **A.2.5.3 Gleitpunktkonstanten**
> [*Kernighan/Ritchie: Programmieren in C (2.Auflage)*]
>
> Eine Gleitpunktkonstante besteht aus einem ganzzahligen Teil, einem Dezimalpunkt, einem Dezimalbruch, dem Zeichen `e` oder `E`, einem ganzzahligen Exponenten mit optionalem Vorzeichen und einem optionalen Typ–Suffix, nämlich einem der Buchstaben `f`, `F`, `l` oder `L`. Ganzzahliger Teil und Dezimalbruch sind Ziffernfolgen. Entweder der Dezimalpunkt oder der Exponent beginnend mit `e`/`E` kann fehlen (aber nicht beide). Der Typ der Konstanten wird durch das Suffix bestimmt; `F` oder `f` machen sie zu `float`, die Suffixe `L` oder `l` machen sie zu `long double`; anderfalls hat sie den Typ `double`.

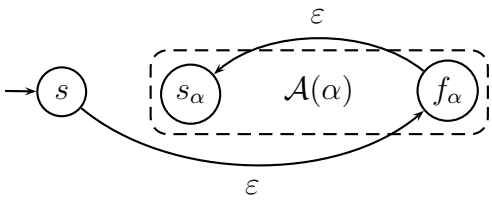
**Aufgabe 12** 8 Punkte

In der Vorlesung wurde bei der Konstruktion nach Thompson (regulärer Ausdruck $\rightsquigarrow$ NFA) auf die Neueinführung von Anfangs- und Endzustand geachtet. Insbesondere beim Stern erscheinen deshalb einige der verwendeten $\varepsilon$-Transitionen/Zustände überflüssig. Wir betrachten nun vereinfachte Konstruktionsverfahren. Welche sind korrekt? Begründen Sie die Korrektheit bzw. widerlegen Sie sie durch ein Gegenbeispiel.

a) $\mathcal{A}(\alpha^*)$ :




b) $\mathcal{A}(\alpha^*)$ :




c) $\mathcal{A}(\alpha^*)$ :