

## 9. Übung „Übersetzerbau“

Abgabe am 20. Juni in der Vorlesung

---

### Aufgabe 28

10 Punkte

In dieser Aufgabe sollen Sie eine einfache Typprüfung implementieren. Wir erweitern MPS zunächst um getypte Deklarationen:

$$\begin{aligned} Decl &\longrightarrow \mathbf{id} : Type, Decl \mid \mathbf{id} : Type \\ Type &\longrightarrow \mathbf{bool} \mid \mathbf{int} \end{aligned}$$

Außerdem unterscheiden wir nun nicht mehr zwischen Bedingungen (*Cond*) und Ausdrücken (*Exp*). Hierdurch vereinfacht sich die Grammatik für *Stm* und *Exp* wie folgt:

$$\begin{aligned} Stm &\rightarrow \mathbf{id} := Exp \mid \mathbf{begin} StmList \mathbf{end} \mid \mathbf{if} Exp \mathbf{then} Stm \mid \mathbf{while} Exp \mathbf{do} Stm \\ Exp &\rightarrow \mathbf{id} \mid \mathbf{numConst} \mid \mathbf{boolConst} \mid (Exp Op Exp) \\ Op &\rightarrow + \mid * \mid <= \mid >= \mid \&\& \mid \parallel \end{aligned}$$

- Definieren Sie algebraische Datentypen (**ASTree**) für die Repräsentation des abstrakten Syntaxbaums eines solchen MPS-Programms.
- Erweitern Sie einen Ihrer Parser für MPS (Top-down oder Bottom-up) um die Generierung des abstrakten Syntaxbaums.
- Implementieren Sie eine Funktion `typeCheck :: ASTree -> Bool`, welche die korrekte Verwendung der Typen **bool** und **int** überprüft.

### Aufgabe 29

10 Punkte

In dieser Aufgabe wollen wir den Zusammenhang zwischen beliebigen zyklfreien attribuierten Grammatiken und Top-Down-Parsern in lazy funktionalen Sprachen untersuchen. Wir betrachten hierzu folgende (zugegeben etwas unsinnige) Programmiersprache:

$$\begin{aligned} Prg &\rightarrow StmList \\ StmList &\rightarrow Stm ; StmList \mid \epsilon \\ Stm &\rightarrow \mathbf{def id} \mid \mathbf{use id} \end{aligned}$$

Die Sprache unterscheidet nur definierende und verwendende Vorkommen von Variablen. Alle verwendeten Variablen müssen in dieser Programmiersprache allerdings auch definiert werden. Dies muss aber nicht vor ihrer Verwendung geschehen.

- a) Definieren Sie für diese Grammatik eine Attributierung, welche diese Eigenschaft überprüft.

Als Einschränkung haben die Nichtterminalsymbole jedoch nur folgende Attribute:

$$\begin{aligned} \text{Syn}(\text{StmList}) &= \text{Syn}(\text{Stm}) = \{b, d\}, \quad \text{Syn}(\text{Prg}) = \{b\} \\ \text{und } \text{Inh}(\text{StmList}) &= \text{Inh}(\text{Stm}) = \{s\} \end{aligned}$$

wobei  $d$  und  $s$  Symboltabellen (also Listen von Variablenbezeichnern) sind und  $b$  vom Typ `Bool` ist. Außerdem habe jedes Terminalsymbol **id** ein synthetisiertes Attribut *name* welches seinen Bezeichner enthalte. Ob in einem Programm alle Bezeichner deklariert sind, soll mit dem Attribut  $b$  des Startsymbols *Prg* angezeigt werden.

- b) Implementieren Sie eine rekursiven Abstiegsparser für diese Grammatik.
- c) In der Vorlesung wurde erläutert, wie L-Attributierungen in rekursiven Abstiegsparsern implementiert werden können. Verwenden Sie genau diese Technik (inherit Attribute als Argumente, synthetisierte Attribute als Rückgabewerte) zur Implementierung der hier entwickelten Attributierung. Entsteht ein ausführbares Programm? Warum/warum nicht?