Lehrstuhl für Programmiersprachen und Übersetzerkonstruktion
CAU Kiel, Institut für Informatik
Prof. Dr. M. Hanus

24118 Kiel
Christan-Albrechts-Platz 4
Tel.: 0431 / 880-7271

6. Übung „Übersetzerbau“

Bearbeitung bis zum 27. Mai 2008

Bitte senden Sie die Programmieraufgabe 21 zusätzlich zur Abgabe in ausgedruckter Form auch per Email an Axel Stronzik (`axs@informatik.uni-kiel.de`)!

Aufgabe 18
Die kontextfreie Grammatik G sei gegeben durch:

$$\begin{aligned} S' &\rightarrow S\$ \\ S &\rightarrow AS \mid b \\ A &\rightarrow SA \mid a \end{aligned}$$

a) Konstruieren Sie (analog zur Vorlesung) einen erfolgreichen Lauf des shift-reduce-Parsers für die Eingabe *abab*.

b) Berechnen Sie den LR(0)-Automaten zu G. Ist $G \in LR(0)$?

Aufgabe 19
Die Grammatik G_{bool} sei gegeben durch

$$\begin{aligned} S &\rightarrow B\$ \\ B &\rightarrow (B \wedge B) \mid \neg B \mid t \mid f \end{aligned}$$

a) Berechnen Sie den LR(0)-Automaten von G_{bool}.

b) Geben Sie die LR(0)–Analysetabelle von G_{bool} an.

c) Bestimmen Sie die Konfigurationsfolge, die der LR(0)–Analyseautomat bei Eingabe des Wortes $((\neg t \wedge f) \wedge (f \wedge t))$ durchläuft.

Aufgabe 20
Zeigen Sie, dass die Klasse REG aller regulären Sprachen schief zur Klasse $\mathfrak{L}(LR(0))$ aller von $LR(0)$–Grammatiken erzeugten Sprachen liegt. Da Sie etwas für Sprachen zeigen sollen, sollten Sie möglichst einfache Sprachen betrachten. Überlegen Sie zunächst, mit welchen Produktionen es bei LR(0) meistens Probleme gibt.

Aufgabe 21
In dieser Aufgabe sollen Sie einen shift-reduce-Parser implementieren. Gehen Sie dazu in folgenden Schritten vor:

a) Formalisieren Sie zunächst, wie und wann der shift-reduce-Parser einen shift- bzw. einen reduce-Schritt durchführt.
Repräsentieren Sie den Keller als ein Wort über Zuständen ($\mathbb{N}^*$). In der Vorlesung wurden auch noch Grammatiksymbole auf den Keller geschrieben. Diese sind für das Arbeiten des Parsers aber überflüssig und können wegfallen.
Um völlig unabhängig von der Grammatik zu sein, erweitern Sie die reduce-Einträge um die Länge der rechten Seite der anzuwendenden Regel und das Nichtterminalsymbol, zu welchem reduziert wird.
Formalisieren Sie auch, in welcher Konfiguration der Parser erfolgreich terminiert.

b) Erweitern Sie den Parser um ein Ausgabeband, auf welchem die Nummer der verwendeten Regel beim reduce-Schritt ausgegeben wird. Liefert Ihr Parser die Rechtsableitung als Ausgabe?

c) Definieren Sie eine polymorphe Datenstruktur `ParsingTabelle a b` zur Darstellung der Parsingtabelle. Die Parsingtabelle soll polymorph bezüglich der gewählten Alphabete sein. `a` repräsentiert die Terminal- und `b` die Nichtterminalsymbole. Da die Parsingtabelle in der Regel nur eine partielle Funktion repräsentiert, also Lücken aufweist, sollten Sie den Haskell-Datentyp `Maybe` verwenden.

d) Definieren Sie eine Funktion `parse :: ParsingTabelle a b -> [a] -> [Int]`. Als Eingabe erhält dieser Parser eine Parsingtabelle und eine Tokenfolge. Als Ausgabe soll der Parser im Erfolgsfall die Ausgabe des Automaten liefern.