Compiler-Compiler-Komponenten I

1) Thema „Scannergenerator“ (4 Personen)

Zweck: Erstellung von Ruby-Scannern aus Jaccie-Scannerdefinitionen, die als Syntaxbäume vorliegen (Analyse der Definition kommt später, wenn auch die Parser implementiert sind, per bootstrap).

Teilaufgaben:

a) Komplette **Syntaxdefinition** (für Scanner und Parser) aufstellen für die Teile „Pattern“ und „Token“ einer Jaccie-Scannerdefinition. [ca 1 Bearbeiter]

b) **Syntaxbäume** zu beliebigen Grammatiken konstruieren, drucken (eingerückt), lesen und schreiben. Dazu (entsprechend dem Composite-Pattern) die drei Klassen Knoten (Attribute @name und @vater), Terminal (ein Attribut für den @quellstring aus dem Scanner) und Nonterminal (Attribute @alternativnr und @soehne, letzteres als Array) verwenden. [ca 1 Bearbeiter]

c) Erstellung von **Ruby-Scannern** zu Jaccie-Scannerdefinitionen, die als Syntaxbäume (im Format von Teilaufgabe b) zu einer Syntaxbeschreibung gemäß Teilaufgabe a vorliegen. [ca 2 Bearbeiter]

   Vorgehensweise:

   - Vorgehen inhaltlich wie in Einarbeitungsaufgabe 3 an den Beispielen aus Teilaufgabe 3.1 erprobt.

   - Zur programmtechnische Umsetzung: die Nonterminal-Knoten des vorliegenden Syntaxbaums um ein weiteres Attribut @rubytext ergänzen und diese Attribute im ganzen Baum rekursiv berechnen. Diese Attrinute enthalten den Ruby-Programmtext des Scanners, der dem Teilbaum entspricht, dessen Wurzel der Knoten ist (evtl. mehr als ein weiteres Attribut erforderlich; daher am besten ein Hash-wertiges Attribut @attribute verwenden – Erläuterung mündlich!!). Dieses Vorgehen entspricht der Verwendung einer Attributgrammatik.

Bemerkung: Die Teilaufgaben sind im Umfang so nicht gleichmäßig verteilt – durch Zusammenarbeit und Unterstützung bei der Doku ausgleichen!

Doku wie in der Einarbeitungsphase!

Bearbeitung durch vier Personen der Scannergruppe der Einarbeitungsphase.

2) Thema „Parsergenerator“ (9 Personen)

Zweck: Zu den Grammatiken aus der Einarbeitungsphase schrittweise wie unten beschrieben verschiedene LR-Parser konstruieren und anwenden.

Teilaufgaben:

a) Zu den Grammatiken aus der Einarbeitungsphase (siehe dort Bestimmung der epsilon-Symbole!) die **first(1)- und follow(1)-Mengen** berechnen, formatiert ausgeben, speichern und laden. Zur Definition siehe Buch S. 70 ff. und Compilerbaubücher, z.B. online das von Dick Grune. [ca 1 Bearbeiter]

b) Zu den Grammatiken berechnen, formatiert ausgeben, speichern und laden:

- den **LR(0)-Automaten** (vgl. S. 156 f. und 165)
- den **LR(1)-Automaten** und den den **SLR(1)-Automaten** (vgl. S. 160 f.)

[ca 2 Bearbeiter]

c) **Aktions- und Übergangstafeln** zu den Automaten aus Teilaufgabe b (die haben alle das gleiche Format!) berechnen, formatiert ausgeben, speichern und laden. Als „Nebenprodukt“: Test auf Konfliktfreikeit (vgl. S. 158 und 162).

[ca 2 Bearbeiter]

d) Einen gemeinsamen **Parserrahmen** („Treiber“) für die Automaten aus Teilaufgabe c in Ruby schreiben (vgl. die Seiten 154 f., 158 f. 160 f.).

[ca 2 Bearbeiter]

e) **Anwendungen:** geeignete Parser erzeugen für:

- die **while-Sprache** (vgl. Einarbeitungsaufgabe)
- die **Scannerdefinitionsgrammatik** (vgl. Aufgabe 1a.)

[ca 2 Bearbeiter]

Doku wie in der Einarbeitungsphase!

Bearbeitung durch die vier Personen der Warshallgruppe der Einarbeitungsphase (für die Teilaufgaben a und b sowie das Anwendungsbeispiel „while-Sprache“), vier Personen der Grammatikgruppe der Einarbeitungsphase (für die Teilaufgaben c und d), sowie ein Mitglied der ScannerGruppe der Einarbeitungsphase (für das Anwendungsbeispiel „Scannerdefinitionssyntax“).

3) Thema „Grammatikeditor“ (4 Personen)

Zweck: Ein Grammatikeditor-Fenster soll das Aufstellen von Grammatiken sicherer gestalten.

Teilaufgaben:

a) Ein **Grammatikeditor**-Fenster nach dem Vorbild der entsprechenden Komponente von SIC implementieren, so dass man Grammatiken erstellen, drucken, speichern, laden und modifizieren kann. Außerdem Zugang zu den in Aufgabe 2 erzeugten Daten (first- und follow-Mengen, verschiedene Automaten, ihre Aktions- und Übergangstafeln) in eigenen Anzeigefenstern (schlichte Textfenster, die die Ausgaben von Aufgabe 2 unverändert und nicht navigierbar anzuzeigen und zu drucken gestatten).

b) Einbau zweier **Spezialfunktionen** für Ausdrucks(teil)-Grammatiken:

- Erzeugung einer Ausdrucksgrammatik wie in der Einarbeitungsaufgabe 2.4.
- Reduktion einer Ausdrucksgrammatik so, dass alle Nonterminale durch ein neues, frei wählbares ersetzt werden und Klammerung entfernt wird.

Zusätzlich ist die Transformation von Syntaxbäumen, die sich beim Parsing ergeben, auf die reduzierte Grammatik zu implementieren (weitere Erläuterungen mündlich!). Reduktions- und Transformationsverfahren wie in der Einarbeitungsphase testen!

Doku wie in der Einarbeitungsphase!

Bearbeitung durch die drei Personen der Plottergruppe der Einarbeitungsphase (für die Teilaufgabe a) und eine Person der Grammatikgruppe der Einarbeitungsphase (für die Teilaufgabe b in Zusammenarbeit mit den anderen).