Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

14. August 2012

Aufgaben zur Klausur **Compilerbau** im SS 2012 (BInf 251, BMInf 252)

Zeit: 90 Minuten erlaubte Hilfsmittel: keine

Bitte tragen Sie Ihre Antworten und fertigen Lösungen ausschließlich an den gekennzeichneten Stellen in das Aufgabenblatt ein. Ist ihre Lösung wesentlich umfangreicher, so überprüfen Sie bitte nochmals Ihren Lösungsweg.

Sollten Unklarheiten oder Mehrdeutigkeiten bei der Aufgabenstellung auftreten, so notieren Sie bitte, wie Sie die Aufgabe interpretiert haben.

Viel Erfolg!

Diese Klausur besteht einschließlich dieses Deckblattes aus 7 Seiten.

Aufgabe 1:

Definieren Sie die Ableitung Δ einer regulären Menge r nach einem Zeichen a:

. . . . . .

. . . . . .

Definieren Sie die Ableitung Δ einer regulären Menge r nach einem Wort w:

. . . . . .

. . . . . .

. . . . . .

Berechnen Sie zu dem regulären Ausdruck $r = (a|bc)*$ über dem Alphabet $\{a, b, c\}$ die Ableitung $\Delta_a(r)$.

Der vereinfachte Ausdrück für die Ableitung:

. . . . . .

. . . . . .

Berechnen Sie zu dem regulären Ausdruck $r = (a|bc)*$ über dem Alphabet $\{a, b, c\}$ die Ableitung $\Delta_{abc}(r)$.

Der vereinfachte Ausdrück für die Ableitung:

. . . . . .

. . . . . .

Aufgabe 2:

1. Warum sind reguläre Ausdrücke ungeeignet, die Syntax einer Programmiersprache vom Umfang von Pascal, C oder Java zu definieren?

   . . . . . .

   . . . . . .

   . . . . . .

2. Warum werden für die lexikalische Analyse reguläre Ausdrücke eingesetzt und nicht kontextfreie Grammatiken?

   . . . . . .

   . . . . . .

   . . . . . .

3. Warum benötigt man zur Überprüfung, ob ein Programm compilierbar ist, neben der Syntaxanalyse noch die Phase der semantischen Analyse?

   . . . . . .

   . . . . . .

   . . . . . .

4. Welche Fehlersituationen werden in der semantischen Analyse erkannt?

   . . . . . .

   . . . . . .

   . . . . . .

Aufgabe 3:

Gegeben sei die folgende kontextfreie Grammatik **G=(T, N, P, S)** mit

T = { Name, Number, String, (,), [,], ., +, − }

N = { args, call, exp, op, prefixexp, var }

S = prefixexp

und den Produktionen **P**:

prefixexp ::= var
prefixexp ::= call
prefixexp ::= (exp)
var ::= Name
var ::= prefixexp [exp]
var ::= prefixexp . Name
call ::= prefixexp args
exp ::= Number
exp ::= String
exp ::= prefixexp
exp ::= exp op exp
exp ::= op exp
op ::= +
op ::= -
args ::= (exp)
args ::= String

Ist diese Grammatik mehrdeutig?

ja ☐ nein ☐

Begründung:

. . . . . .

. . . . . .

. . . . . .

Berechnen Sie die FIRST-Mengen für die Symbole prefixexp, var, call und exp .

prefixexp . . . . . .

var . . . . . .

call . . . . . .

exp . . . . . .

An den FIRST-Mengen erkennt man, dass diese Grammatik keine LL(1) Grammatik ist.
Schreiben Sie die Regeln für exp so um, dass für das Symbol exp das LL-Parsen möglich wird:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Diese reicht noch nicht, um die LL-Eigenschaft zu bekommen. Fassen sie hierfür die Regeln für prefixexp, var und call so zusammen, dass die Symbole var und call eliminiert werden (Nebenrechnung).

Die neu entstandenen Regeln für prefixexp müssen dann transformiert werden, damit diese ebenfalls die LL(1)-Eigenschaft nicht zerstören.

Die neuen Regeln für prefixexp:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Aufgabe 4:

Skizzieren Sie, welcher Assembler-Code für einen von–Neumann–Rechner für die folgende Funktion sinnvollerweise erzeugt wird. Verwenden Sie für die Befehle die in der Vorlesung vorgestellte virtuelle Maschine und deren Instruktionssatz.

function f(n : **int**) : **int**
  if n $<$ 1
  then n
  else f(n $-$ 1) $*$ f(n $-$ 2)

Verwenden Sie in den Instruktionen die Namen der Variablen als symbolische Adressen für die zugehörigen Speicheradressen.

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .