Prüfung Compiler

Datum	:	27.01.2015, 10:30 Uhr
Bearbeitungszeit	:	90 Minuten
Prüfer	:	Prof. Dr. Oliver Braun
Hilfmittel	:	Keine
Erreichbare Punkte	:	90

Name: _____

Vorname: _____

Matrikelnummer: _____ Studiengruppe: _____

Hörsaal: _____ Platz Nr.: _____

Unterschrift: _____

Bitte kontrollieren Sie, ob Sie eine vollständige Angabe mit 6 Aufgaben auf 9 Seiten erhalten haben.

Aufgabe	1	2	3	4	5	6	Summe
max. Punkte	12	16	18	24	10	10	90

Anmerkungen:

- Schreiben Sie die Lösungen in die dafür vorgesehenen Kästchen. Sollte Ihnen der Platz dabei nicht reichen, benutzen Sie die Rückseite **und vermerken Sie das im dazugehörigen Kästchen!**

Aufgabe 1 (12 Punkte)

Gegeben sei folgender Java-ähnlicher Quellcode:

```
1  int x = 12;
2  int y = 7;
3  while (y<12) {
4      int a = 10;
5      while (a>0) a--;
6      if (y!=x)
7          a = x-- + --y;
8      else
9          a = 5
10     y--;
11 }
```

(a) Geben Sie den Kontrollflußgraph (*control-flow graph*) für obigen Code an. (6)

(b) Geben Sie den Abhängigkeitsgraph (*dependence graph*) für obigen Code an. (6)

Aufgabe 2 (16 Punkte)

Gegeben sei folgender Code in einer Java-ähnlichen Programmiersprache:

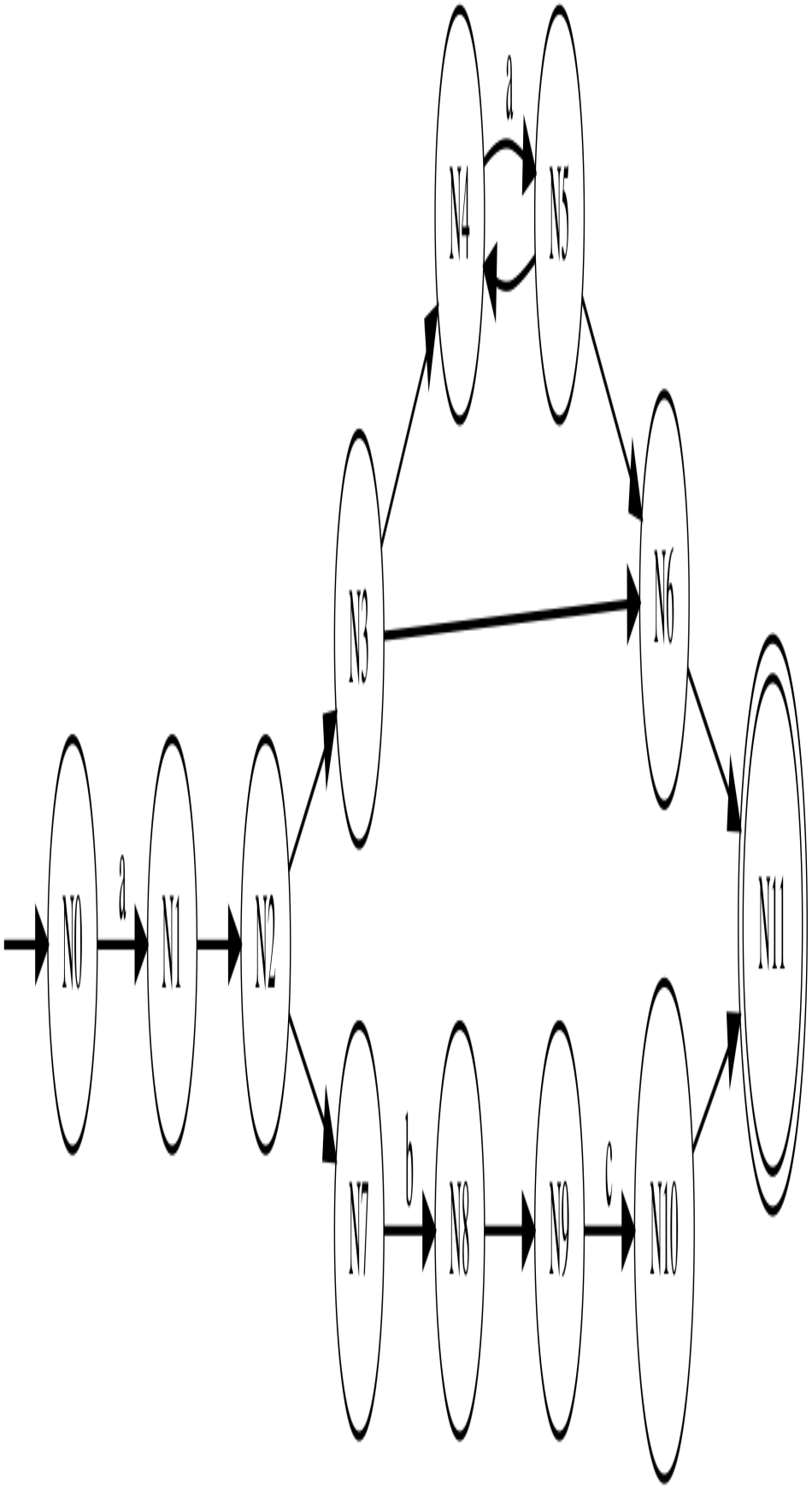
```
for (i = 0 ; i < 10 ; i++) {
    if (i % 2)
        continue;
    else
        x = x + i;
}
```

(a) Geben Sie den Token-Strom an, den ein Scanner hier erkennen muss. Sie können die Token entweder als Haskell-Token oder als Tupel bestehend aus dem Token und der syntaktischen Kategorie angeben, also z.B. entweder If oder (if, Schlüsselwort). (8)

(b) Geben Sie den den Code repräsentierenden Syntax-**DAG** an, den der Parser aus dem Tokenstrom als sinnvolle Zwischenrepräsentation generiert. (8)

Aufgabe 3 (18 Punkte)

Gegeben sei folgender NFA:


N0
a
N1
N2
N3
N4
a
N5
N6
N7
b
N8
N9
c
N10
N11

Erzeugen Sie mit Hilfe der **Teilmengenkonstruktion** (*subset construction*) daraus einen DFA.

Geben Sie alle ermittelten Teilmengen **und** den resultierenden DFA an.

Aufgabe 4 (24 Punkte)

Geben Sie für die folgenden regulären Ausdrücke jeweils einen **deterministischen endlichen Automat**, der die gleichen Zeichenfolgen akzeptiert, als Zustandsübergangsdiagramm an. Sie brauchen die Zustände nicht benennen, es reicht wenn Sie kleine Kreise zeichnen. Sie brauchen den DFA nicht formal herleiten bzw. konstruieren.

(a) $ke^*w|coo|um^*$ (6)

(b) $(w|wr|wwr|r)x^*xw$ (8)

(c) $a(a|c)a|b^*b(b|ca)$ (10)

Aufgabe 5 (10 Punkte)

Eliminieren Sie in der folgenden Grammatik die Linksrekursion.

0	*Goal*	$\mapsto$	*Expr*
1	*Expr*	$\mapsto$	*Expr* + *Term*
2		|	*Expr* - *Term*
3		|	*Term*
4	*Term*	$\mapsto$	*Term* * *Factor*
5		|	*Term* / *Factor*
6		|	*Factor*
7	Factor	$\mapsto$	(*Expr*)
8		|	`num`
9		|	`name`

Sie müssen nur die geänderten/neuen Regeln angeben.

Aufgabe 6 (10 Punkte)

Gegeben sei der Tokentyp

```
data Token = NatNum Integer | Add | Mult
```

Die zugrunde liegende Expressionlanguage verlangt, dass alle Operatoren in polnischer Notation, d.h. Präfix, geschrieben werden müssen. Der Vorteil ist, dass wir damit ohne Klammern auskommen.

Gegeben ist außerdem folgende `parse'`-Funktion:

```
1  parse' :: [Token] -> (Maybe AST, [Token])
2  parse' (NatNum i    :ts) = (Just (I i),   ts)
3  parse' (Add         :ts) =
4    case parse' ts of
5      (Just fstOp, rest) -> case parse' rest of
6        (Just sndOp, rest') -> (Just (Plus fstOp sndOp), rest')
7        _ -> (Nothing, ts)
8      _ -> (Nothing, ts)
9  parse' (Mult         :ts) =
10   case parse' ts of
11     (Just fstOp, rest) -> case parse' rest of
12       (Just sndOp, rest') -> (Just (Times fstOp sndOp), rest')
13       _ -> (Nothing, ts)
14     _ -> (Nothing, ts)
```

(a) Geben Sie den Teil des `AST`-Datentyps an, der in dem obigen Code benötigt wird. (`data AST = ...`) (3)

(b) Was ist das Ergebnis des Aufrufs (3)

```
parse' [Add, NatNum 123, Mult, NatNum 456, NatNum 789]
```

(c) Was ist das Ergebnis des Aufrufs (4)

```
parse' [Add, NatNum 123, NatNum 456, NatNum 789]
```