

Programmierkurs Java

Dr. Dietrich Boles

Aufgaben zu UE3-Syntaxdiagramme (Stand 05.11.2010)

Aufgabe 1:

Entwickeln Sie Regeln zur Übersetzung von EBNF in Syntaxdiagramme.

Aufgabe 2:

Eine Zahl ist entweder eine positive Zahl oder ein Minuszeichen gefolgt von einer positiven Zahl oder das Zeichen Null. Eine positive Zahl ist eine Ziffer außer Null gefolgt von einer optionalen Ziffernfolge. Eine optionale Ziffernfolge ist eine Ziffer gefolgt von einer optionalen Ziffernfolge oder leer.

- (a) Überführen Sie die Aussage in eine BNF
- (b) Überführen Sie diese Aussage in eine EBNF

Aufgabe 3:

Gegeben sei folgende EBNF für Gleitkomma-Literale in Java:

```
<FloatPointLit> ::= <Digits> \. [ <Digits> ] [ <ExpPart> ] [ <FloatTypeSuffix> ]
                  | \. <Digits> [ <ExpPart> ] [ <FloatTypeSuffix> ]
                  | <Digits> <ExpPart> [ <FloatTypeSuffix> ]
                  | <Digits> [ <ExpPart> ] <FloatTypeSuffix>

<Digits> ::= <Digit> { <Digits> }
<Digit> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
<ExpPart> ::= <ExpIndicator> <SignedInteger>
<ExpIndicator> ::= 'e' | 'E'
<SignedInteger> ::= [ <Sign> ] <Digits>
<Sign> ::= '+' | '-'
<FloatTypeSuffix> ::= 'f' | 'F' | 'd' | 'D'
```

[...] bedeutet: Symbole oder Symbolfolgen innerhalb der Klammern können auch weggelassen werden

{...} bedeutet: Symbole oder Symbolfolgen innerhalb der Klammern können beliebig oft wiederholt oder auch ganz weggelassen werden

... | ... bedeutet: genau ein alternatives Symbol oder eine alternative Symbolfolge innerhalb der Klammern muss auftreten

<Zeichenfolge>: Nichtterminal

'Zeichenfolge': Terminal

Teilaufgabe (a):

Formen Sie die obige EBNF in ein äquivalentes Syntaxdiagramm um!

Teilaufgabe (b):

Welche der folgenden Zeichenketten sind bezüglich der obigen EBNF syntaktisch korrekt, welche nicht (Begründen Sie Ihre Entscheidung!):

1. 4711.08155e23D
2. 0815.f
3. .ef
4. +4.5
5. 3456.7<ExpPart>F
6. 6.4E-23f
7. 78.2f
8. 78g.2f
9. 99999999999.999999999e+999999999999999
10. 6767.7676.7e56

Aufgabe 4:

Entwickeln Sie ein Syntaxdiagramm (bzw. alternativ eine EBNF) für EBNF-Regeln (d.h. beschreiben Sie, wie EBNF-Regeln syntaktisch aufgebaut sind)!

Aufgabe 5:

Gegeben sei folgende EBNF:

```
<Start> ::= <Digits> \. [ <sVocal> ] <bVocal>
          | \. <Digits> { <bVocal> }
          | <sVocal> { <bVocal> } <Digit>

<Digits> ::= <Digit> { <Digit> }

<Digit> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

<sVocal> ::= 'a' | 'e' | 'i' | 'o' | 'u'

<bVocal> ::= 'A' | 'E' | 'I' | 'O' | 'U'
```

1. 4711.E
2. .AE
3. i56
4. 88.eeAA
5. eAEE7
6. 815+aA

Aufgabe 6:

Entwickeln Sie ein Syntaxdiagramm (bzw. alternativ eine EBNF) für die Sprache $L = \{(a^2b)^n(a^2b)^m \mid n, m \text{ sind natürliche Zahlen oder Null}\}$

Aufgabe 7:

Gegeben sei folgende EBNF:

```

<InterfaceDecl> ::= [ <InterfaceMod> ] 'interface' <Identifier>
                  [ 'extends' <Identifier> { ',' <Identifier> } ]
                  ( <InterfaceBody> | ';' )

<InterfaceMod> ::= ( 'public' | 'protected' | 'private' )

<Identifier> ::= <Letter> { ( <Letter> | <Digit> ) }

<InterfaceBody> ::= '{' { <InterfaceMemD> } '}'

<InterfaceMemD> ::= ( 'method' | 'constant' | 'class' )

<Letter> ::= ( 'a' | 'b' | '$' )

<Digit> ::= ( '0' | '1' )
  
```

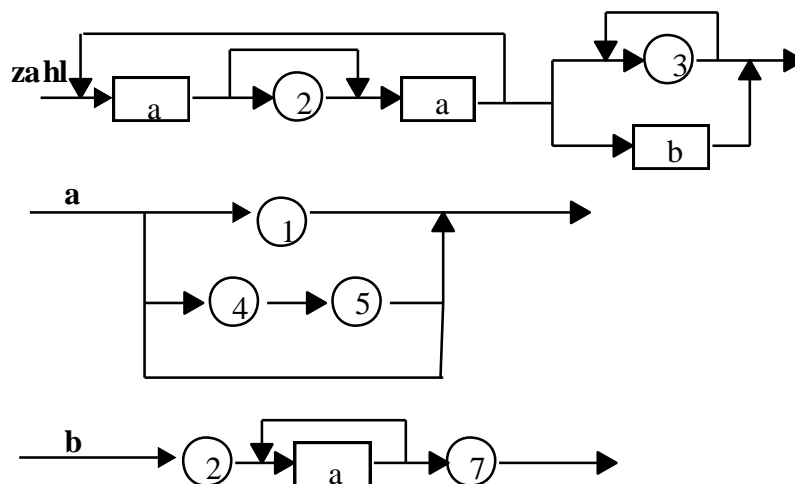
1. interface ablalal;
2. public interface aaa extends bbb, ab\$ { method method }
3. protected interface 0\$ab extends aaa method constant
4. interface aaa, bbb extends aaa, bbb { aaa }
5. InterfaceMod interface extends Identifier ;
6. private public interface extends aaa , { class extends bbb }

Aufgabe 8:

Ist es möglich, ein Syntaxdiagramm (bzw. alternativ eine EBNF) für die Sprache $L = \{(a^n d^2 b^m c^3)^p \mid n, m \text{ sind natürliche Zahlen oder Null, } p \text{ ist natürliche Zahl größer Null}\}$ zu entwickeln?

Aufgabe 9:

Gegeben sei folgendes Syntaxdiagramm:



Welche der folgenden Zeichenketten sind syntaktisch korrekt, welche nicht:

- 12451333
- 12459333
- 12a3
- 145245
- 2217

- 454545221452127
- 123453
- 4545217

Begründen Sie Ihre Antwort!

Aufgabe 10

Entwickeln Sie eine EBNF für die Sprache
 $L = \{x^2y^n z^m (cd)^3 \mid n, m \text{ sind natürliche Zahlen oder Null}\}$

Aufgabe 11:

Ist es möglich für folgende Sprache L eine EBNF zu entwickeln?
 $L = \{(x^n y^m)^p \mid n, m, p \text{ sind natürliche Zahlen größer oder gleich Null}\}$

Aufgabe 12:

Ist es möglich für folgende Sprache L eine EBNF zu entwickeln?
 $L = \{a^n b^n c^n \mid n \text{ ist natürliche Zahl oder Null}\}$

Aufgabe 13:

Die Menge Q der Brüche besteht aus allen Worten, die aus einer ganzen Zahl, gefolgt von einem Schrägstrich (/) und einer ganzen Zahl ungleich 0 bestehen.

Geben Sie eine EBNF für die Menge Q an.

Aufgabe 14:

Gegeben sei die folgende Grammatik in EBNF (die Syntax entspricht übrigens in etwa der Syntax der funktionalen Programmiersprache *Scheme*):

```

<Ausdruck> ::= <Atom> | <Liste>
<Atom> ::= <Zahl> | <Zeichenkette>
<Liste> ::= , ( ' <Ausdruck-Folge> , ) '
<Ausdruck-Folge> ::= <Ausdruck> <Ausdruck-Folge> | ε
<Zahl> ::= <Ziffer> { <Ziffer> }
<Ziffer> ::= , 0 ' | , 1 ' | , 2 ' | , 3 ' | , 4 '
<Zeichenkette> ::= <Zeichen> <Zeichenkette> | ε
<Zeichen> ::= , i ' | , f ' | , / ' | , + ' | , = ' | , a '

```

Welche der folgenden Sätze/Programme sind korrekt, welche nicht! Begründen Sie Ihre Entscheidung!

- (1) (/ 22 (+ 41 3))
- (2) (if (= a 0)
 - 0
 - (/ 1 a))
- (3) (+ 41 3) (+ 3 41)
- (4) (/ 1 2 3 (4 3 (* 2 1))
- (5) (* 2.3 4.6)
- (6) (+ Ziffer Zeichenkette)
- (7) (fifi (iffi iffi) ifif)

Aufgabe 15:

Die Sprache Mini-Pascal (aus dem Buch "Vom Problem zum Programm" von Herbert Klaeren, erschienen beim Teubner Verlag) ist in etwa durch die folgende EBNF definiert:

```

<program> := "program" <ident> ";" <block> ".".
<block> := [<vardecl>] "begin" <statement> {";" <statement>} "end".
<vardecl> := "var" <ident> {"," <ident>} ":" "Integer" ";".
<statement> := <ident> "!=" <expr>
              | "begin" <statement> {";" <statement>} "end"
              | "if" <condition> "then" <statement>
              | "while" <condition> "do" <statement>
              | "read" <ident>
              | "write" <ident>.
<condition> := <expr> ("=" | "<" | "<" | ">" | "<=" | ">=") <expr>.
<expr> := <term> {"+" | "-"} <term>}.
<term> := <factor> {"*" | "/" } <factor>}.
<factor> := <ident> | <number> | "(" expr ")".
<ident> := <letter> {<letter>}
<number> := <digit> {<digit>}
<letter> := "a" | "b" | "c"
<digit> := "0" | "1"

```

Konstruieren Sie syntaktisch korrekte Mini-Pascal-Programme.