Christian Ikenmeyer, Peter Pfahler, Florian Schoppmann, Adrian Slowik, Patrick van Lück, Björn Zeiger

Vorlesung Grundlagen der Programmiersprachen Sommer 2004 - Lösung zu Blatt 2

Lösung zu Aufgabe 3

Diese Aufgabe wird zu einem späteren Zeitpunkt erneut behandelt.

Lösung zu Aufgabe 4

Die folgenden Aussagen aus dem Java Reference Manual sollten den vier Ebenen der Spracheigenschaften (Folie 116) zugeordnet werden:

1. *A block is a sequence of statements and local variable declaration statements within braces.*
   (b) Syntax: beschreibt die Struktur eines Blocks in Java.
2. *A return statement with an expression must be contained in a method declaration that is declared to return a value or a compile-time error occurs.*
   (c) statische Semantik: beschreibt Anforderungen an die Signatur (den Ergebnistyp) der umgebenden Methodendefinition; zur Übersetzungszeit überprüfbar.
3. *If execution of the try block completes normally, then no further action is taken and the try statement completes normally.*
   (d) dynamische Semantik: beschreibt die Wirkung eines try-Blocks bei der Ausführung eines Java-Programms.
4. *The boolean type has two values, represented by the literals true and false, formed from ASCII letters.*
   (a) Grundsymbole: beschreibt die Notation der beiden Literale des Typs boolean in Java.
5. *A widening conversion of a signed integer value to an integral type T simply sign-extends the two's-complement representation of the integer value to fill the wider format.*
   (d) dynamische Semantik: beschreibt die Wirkung einer bestimmten Art von Typumwandlung bei der Ausführung eines Java-Programms.
6. It is a compile-time error for a line terminator to appear after the opening " and before the closing matching ".
   (a) Grundsymbole: beschreibt eine Einschränkung für die Notation von Zeichenreihen-Literalen in Java.
7. The type of the operand expression of the unary - operator must be a primitive numeric type.
   (c) statische Semantik: beschreibt Anforderungen an den statischen Typ der Operanden eines bestimmten Java-Operators; zur Übersetzungszeit überprüfbar.
8. An array type is written as the name of an element type followed by some number of empty pairs of square brackets [].
   (b) Syntax: beschreibt die Struktur (nicht einfach nur die Notation) einer Typangabe für Reihungstypen in Java.

Lösung zu Aufgabe 5

```
p1:   prog      ::= 'main' stmtblock
p2:   stmtblock ::= '{' stmts '}'
p3:   stmtblock ::= stmt
p4:   stmts     ::= stmts stmt
p5:   stmts     ::= stmt
p6:   stmt      ::= app_id '=' expr ';'
p7:   stmt      ::= 'if' '(' expr ')' stmtblock 'else' stmtblock
p8:   stmt      ::= 'while' '(' expr ')' stmtblock
p9:   stmt      ::= 'if' '(' expr ')' stmtblock
p10:  stmt      ::= 'input' '(' app_id ')' ';'
p11:  stmt      ::= 'output' '(' app_id ')' ';'
p12:  app_id    ::= identifier
p13:  expr      ::= call
p14:  expr      ::= integer
p15:  expr      ::= app_id
```

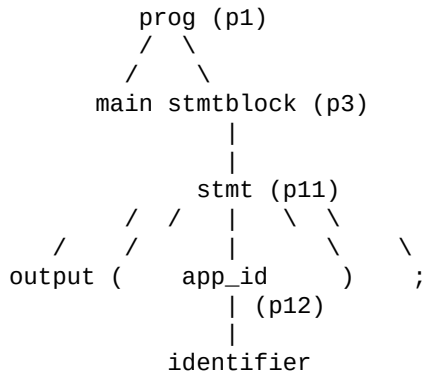
```
p16:  call      ::= app_id '(' params ')'
p17:  params    ::= params ',' params
p18:  params    ::= param
p19:  param     ::= expr
```

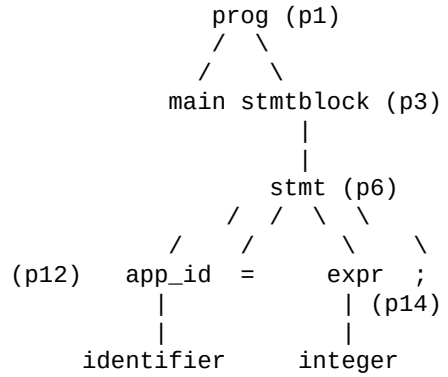
a) Geben Sie die Terminalsymbole und Nichtterminalsymbole der Grammatik an.

```
N = { prog, stmtblock, stmts, stmt, app_id, expr, call, params, param }

T = { 'main', '{', '}', '=', ';', 'if', '(', ')', 'else', 'while', 'input','output', identifier, integer, ',' }
```

b) Wie sieht der kleinste Ableitungsbaum (in Anzahl Knoten) zu dieser Grammatik aus? Mit 4 bzw. 5 Ableitungsschritten erhält man z.B.:





c) Herr D. E. Tektiv stellt beim Lesen der Grammatik fest, dass die Grammatik mehrdeutig ist. Markieren Sie alle Produktionen, die Mehrdeutigkeiten herbeiführen. Schlagen Sie geeignete Maßnahmen vor, mit denen man die Mehrdeutigkeiten entfernen kann, ohne die Sprache zu verändern.

Teil a)
Mehrdeutigkeit in

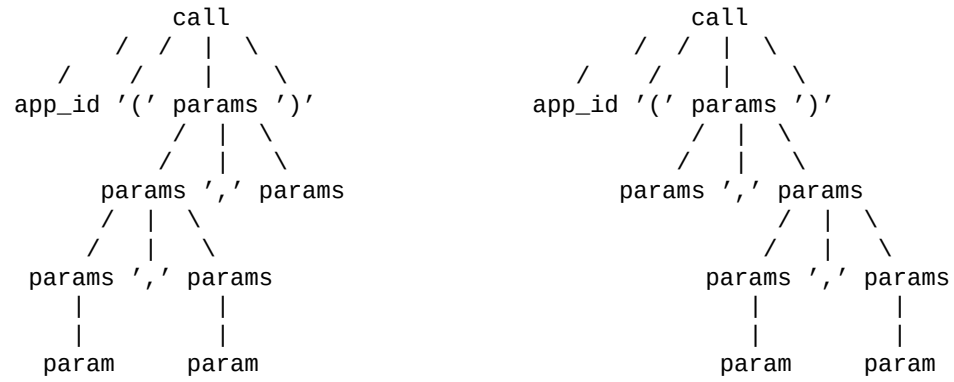
```
p16: call ::= app_id '(' params ')'
p17: params ::= params ',' params
p18: params ::= param
p19: param ::= expr
```

Assoziativität des `','` ist nicht bestimmt.

Es gibt einen Satz, zu dem es zwei verschiedene Ableitungsbäume gibt. Es genügt, ein Beispiel anzugeben: Dazu eignet sich jeder Satz, der einen Aufruf der Form enthält

```
f (a, b, c)
```

Wir geben zwei verschiedene Ausschnitte aus Ableitungsbäumen dazu an:



Abhilfe wie in eindeutigen Ausdrucksgrammatiken:

```
p16:  call ::= app_id '(' params ')'
p17': params ::= params ',' param
p18:  params ::= param
p19:  param ::= expr
```

Teil b)
Mehrdeutigkeit in

```
p7: stmt ::= 'if' '(' expr ')' stmtblock 'else' stmtblock
p9: stmt ::= 'if' '(' expr ')' stmtblock
```

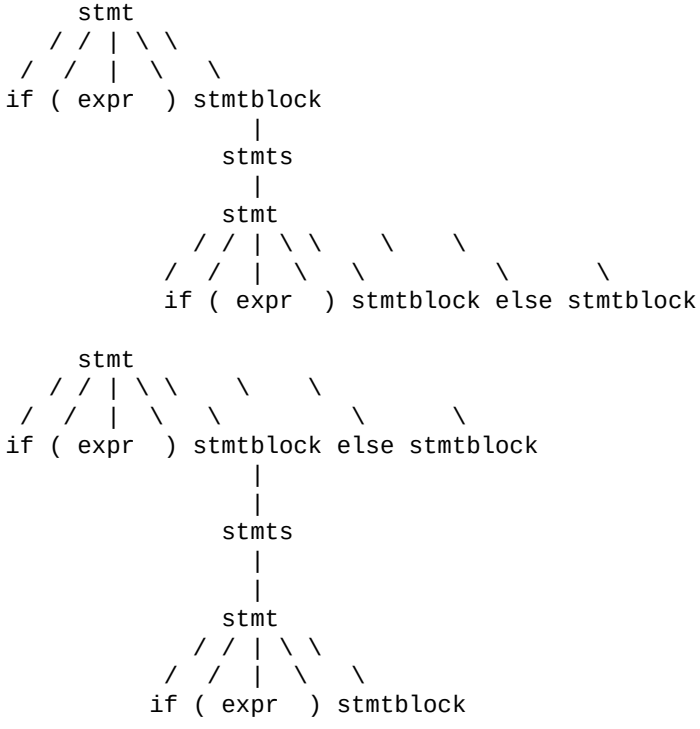
dangling else; siehe Folie GdP-2-15

Es gibt einen Satz, zu dem es zwei verschiedene Ableitungsbäume gibt.

Es genügt, ein Beispiel anzugeben:
Geignet ist jeder Satz, der folgendermaßen geschachtelte `if`-Konstrukte enthält

```
if (expr) stmtblock else stmtblock
```

Wir geben zwei verschiedene Ausschnitte aus Ableitungsbäumen dazu an:



Abhilfe erfordert Einführen eines neuen Nichtterminals wie `stmtBlockNoShortIf` und zusätzlicher Produktionen mit diesem statt mit `stmtblock`. (Siehe Java Manual).

Stattdessen meist: verbale Erklärung und Auflösen mit Übersetzertechnik

d) Geben Sie die Ableitungsschritte an, mit denen der folgende Satz aus dem Startsymbol abgeleitet werden kann:

```
main {
    input(value);
    if (value)
        output(log(value));
}
```

Dieser Satz ist mit der Grammatik nich ableitbar. Ersetzt man `p11` durch `p11': stmt ::= 'output' '(' expr ') ';'`, dann:

```
       prog
p1  => main stmtblock
p2  => main { stmts }
p3  => main { stmts stmt }
p4  => main { stmt stmt }
p10 => main { input ( app_id ); stmt }
p12 => main { input ( identifier); stmt }
```

Für das 2. `stmt`:

```
        stmt
p9   => if (expr) stmtblock
p15  => if (app_id) stmtblock
p12  => if (identifier) stmtblock
p3   => if (identifier) stmt
p11' => if (identifier) output ( expr )
p13  => if (identifier) output ( call )
p16  => if (identifier) output ( app_id ( params ) )
p12  => if (identifier) output ( identifier ( params ) )
p18  => if (identifier) output ( identifier ( param ) )
p19  => if (identifier) output ( identifier ( expr ) )
p15  => if (identifier) output ( identifier ( app_id) )
p12  => if (identifier) output ( identifier (identifier));
```