Übung zur Vorlesung Compiler 1: Grundlagen

Prof. Dr. Andreas Koch
Jens Huthmann

TECHNISCHE UNIVERSITÄT DARMSTADT

Wintersemester 12/13
Aufgabenblatt 1 - Lösungsvorschlag

Abgabemodalitäten

Die Aufgabe sind in 2er Gruppen zu bearbeiten. Ihre Lösungen reichen Sie als PDF per E-Mail bis zum 23.11.2012 um 23:59 MET an der Adresse oc@esa.informatik.tu-darmstadt.de ein. Die E-Mail muss als Betreff "Compiler 1 Aufgabe 1" haben. Geben Sie in Ihrem Lösungsdokument den Namen und die Matrikelnummer aller Gruppenmitglieder an.

Aufgabe 1.1 Triangle zu AST — 10 Punkte

Zeichnen Sie für das folgende Triangle Programm den abstrakten Syntaxbaum. Halten Sie sich dabei an den Regeln auf auf Seite 109 und 110 im Buch. Zum Zeichnen können Sie zum Beispiel Visio verwenden, welches Sie im MSDNAA bekommen können.

```
if (x < 40)
        x := x * 2
else
        x := x + 1
```

Aufgabe 1.1 Lösung

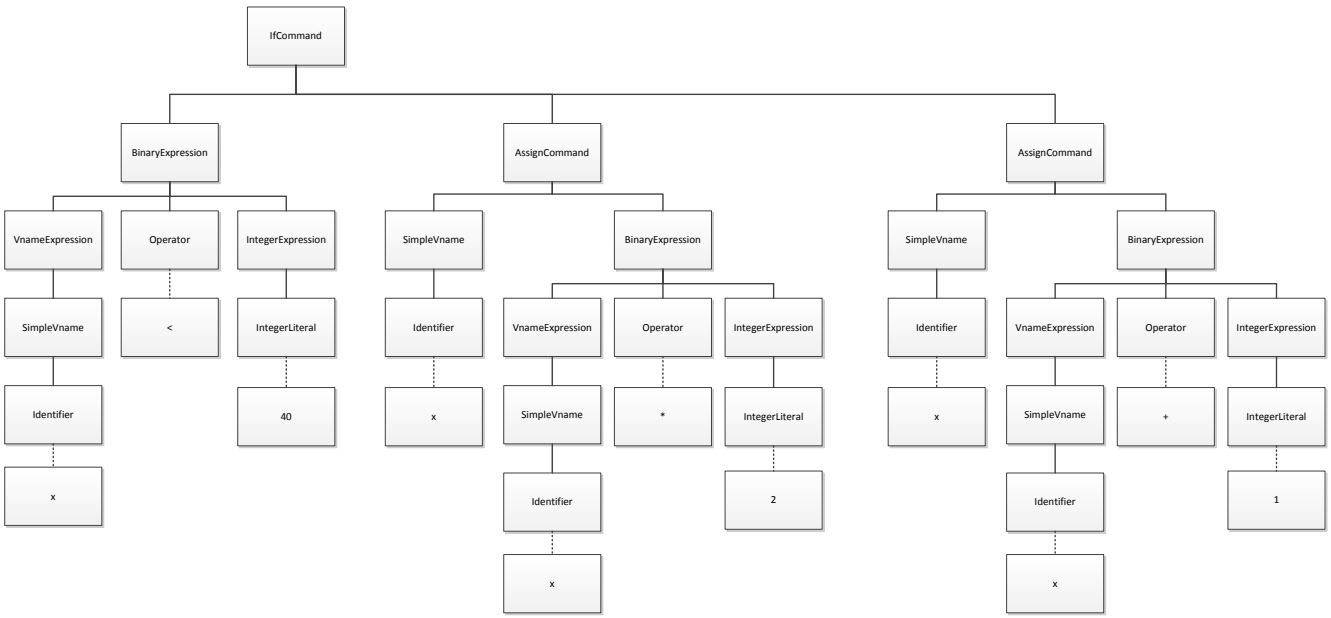


Abbildung 1: Lösung der Aufgabe 1.1

Aufgabe 1.2 AST zu Triangle — 10 Punkte

Geben Sie für den abstrakten Syntaxbaum auf Seite 6, 7 und 8 das zugehörige Triangle Programm an. Die AST Regeln auf Seite 109 und 110 im Buch wurden um eine Regel für Array erweitert. Diese Regel gehört in die Gruppe der V-name ASTs. Zusätzlich gibt es noch einen SkipCommand welcher einem leeren Befehl entspricht. Ein Beispiel hierzu gibt es auf Seite 21 des Buches (Example 1.9).

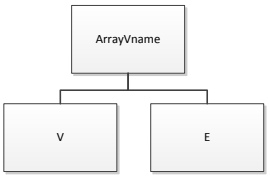


Abbildung 2: Regel für Arrays

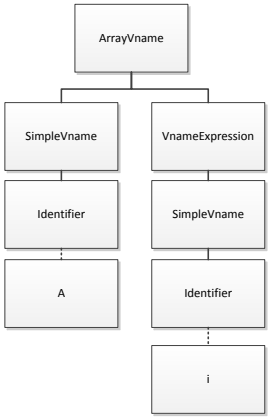


Abbildung 3: Beispiel

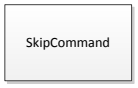


Abbildung 4: Leerer Befehl

Aufgabe 1.2 Lösung

```
let
  var i: Integer;
  var j: Integer;
  var buf : Integer;
  var x : Integer
in begin
  i := 0;

  while (i < n) do begin
    x := n - i - 2;
    j := 0;
    while (j <= x) do begin
      if (val[j] > val[j+1]) then
        begin
          buf := val[j];
          val[j] := val[j+1];
          val[j+1] := buf;
        end
      else;
      j := j + 1;
    end;
    i := i + 1;
  end

end;
```

Aufgabe 1.3 LL(1) Grammatiken — 15 Punkte

Welche der gegebenen Grammatiken sind durch einen LL(1) Parser zu verarbeiten? Falls eine Grammatik dies nicht ist, so formen Sie diese entweder um, so dass sie verarbeitet werden kann, oder begründen Sie, warum eine Umformung gegebenenfalls nicht möglich ist.

a) $G = (\{S, A, B\}, \{x, y, z, 0, 1\}, P, S), P:$
$S ::= A \mid B$
$A ::= x\ A\ y \mid 0$
$B ::= x\ B\ z \mid 1$

b) $G = (\{A, B, C, D\}, \{\text{if}(,), \text{else}, \text{fi}, \text{true}, \text{false}, a, b, c\}, P, A), P:$
$A ::= \text{if}\ (\ B\ )\ A\ C \mid \text{D}$

$B ::= \text{true} \mid \text{false}$
$C ::= \text{else } A \text{ fi} \mid \text{fi}$
$D ::= a \mid b \mid c$

c) $G = (\{S, X, Y\}, \{0, 1, 2, a, b\}, P, S), P:$
$S ::= X\ 0 \mid Y\ 1 \mid 2 \mid S\ S$
$X ::= Y$
$Y ::= a \mid b$

Aufgabe 1.3 Lösung

a) Diese Grammatik ist durch einen LL(1) Parser nicht zu verarbeiten.

b) Die vorliegende Grammatik kann durch einen LL(1) Parser verabeitet werden.

c) Diese Grammatik ist in folgendem, umgeformten Zustand LL(1) konform.
$G = (\{S, Y\}, \{0, 1, 2, a, b\}, P, S), P:$
$S ::= (Y\ (0 \mid 1) \mid 2)+$
$Y ::= a|b$

Aufgabe 1.4 LL(k) Grammatiken — 10 Punkte

Bestimmen Sie unter Verwendung der Zerlegungsregeln von Foliensatz 2, Folie 51 ob die gegebene Grammatik LL(1) ist.

$G = (\{S, B\}, \{a, b\}, P, S), P:$
$S ::= aXab \mid bXbb$
$B ::= a \mid \epsilon$

Aufgabe 1.4 Lösung

Zum Lösen dieser Aufgabe ist ein Test auf die Anwendbarkeit der Zerlegungsregeln von Foliensatz 2, Folie 51 notwendig.

Man sieht sofort das die Grammatik keine Regel der Form $X*$ enthält. Daher kann durch diese auch nicht die Bedingungen verletzt werden.

Die Grammatik enthält aber eine Regel der Form $X|Y$ mit der Produktion $X ::= a \mid \epsilon$. Es ist also notwendig zu überprüfen ob $\text{starters}[[X]] \cap (\text{starters}[[Y]] \cup \text{follow}[[X|Y]]) = \emptyset$ verletzt wird.

$X = a$
$Y = \epsilon$
$\text{starters}_1[[a]] = \{a\}$
$\text{starters}_1[[\epsilon]] = \emptyset$
$\text{follow}_1[[a|\epsilon]] = \{a, b\}$
$\text{starters}_1[[\epsilon]] \cup \text{follow}[[y|\epsilon]] = \{a, b\}$
$\text{starters}_1[[a]] \cap \{a, b\} = \{a\} \neq \emptyset$

Die Regel wird also für $k = 1$ verletzt.

Aufgabe 1.5 Parser — 15 Punkte

Geben Sie die Namen der Java Klassen des AST für die gegebene Grammatik an. Unterstreichen sie hierbei die Klassen welche die Schreibweise des Tokens benötigen. Schreiben Sie dann die benötigten parseN-Methoden. Halten Sie sich bei Ihrer Umsetzung an das Beispiel in den Folien bzw. des Buches (Seite 95, Schritt 2). Jedes Terminalsymbol außer denen aus der Produktion ID sind als Schlüsselwort zu betrachten.

Es ist nicht notwendig die AST Klassen zu implementieren. Substitution ist nicht erlaubt.

$G = (\{A, B, C, D\}, \{\text{foo}(,), \text{bar}, \text{true}, \text{false}, a, b, c\}, P, A), P:$
$A ::= \text{foo}\ (\ B\ )\ A\ C \mid \text{D}$
$B ::= \text{true} \mid \text{false}$
$C ::= \text{bar}$
$D ::= 1 \mid 2 \mid 3$

Aufgabe 1.5 Lösung

AST Klassen: A, foo, B, true, false, bar, C, D

```
private void parseA () {
  switch (currentToken.kind) {

  case Token.foo:
    {
      acceptIt();
      acceptToken(Token.openBrace);
      parseB();
      acceptToken(Token.closeBrace);
      parseA();
      parseC();
    }
    break;

  case Token.D:
    {
      acceptIt();
    }
    break;
  }
}

private void parseC() {
  switch (currentToken.kind) {

  case Token.bar:
    {
      acceptIt();
    }
    break;
  }
}

private void parseB() {
  switch (currentToken.kind) {

  case Token.true:
    {
      acceptIt();
    }
    break;

  case Token.false:
    {
      acceptIt();
    }
    break;
  }
}
```

Aufgabe 1.6 Scanner — 15 Punkte

Nehmen Sie an, dass die lexikalische Grammatik von Mini-Triangle um hexadezimale Literale erweitert wird. Geben Sie hierzu analog zu den Verfahren im Kapitel 4.5 ab Seite 118 im Buch einen passenden Scanner an. Hierbei reicht es aus die ”scanToken” Methode zu schreiben. Die Fälle für einzelnen Buchstaben oder Ziffern die keine Sonderfälle betreffen

können sie mit case 'a' ... case 'z' zusammenfassen. Beachten Sie hierbei dass es passieren kann das Sie von dem Verfahren für einige Regeln leicht abweichen müssen.

Digit_0 ::= 1 ... 9
Digit_8 ::= 0 ... 7
Digit ::= 0 ... 9

Token ::= Identifier | Operator | IntegerLiteral | HexLiteral | ...
Identifier ::= Letter (Letter | Digit)*
IntegerLiteral ::= 0 | (Digit_0 (Digit)*)
OctalLiteral ::= 0 (Digit_8)+

Nennen Sie anschliesend das wesentliche Problem dieser Grammatik welches Sie zur Abweichung von dem Verfahren gezwungen hat.

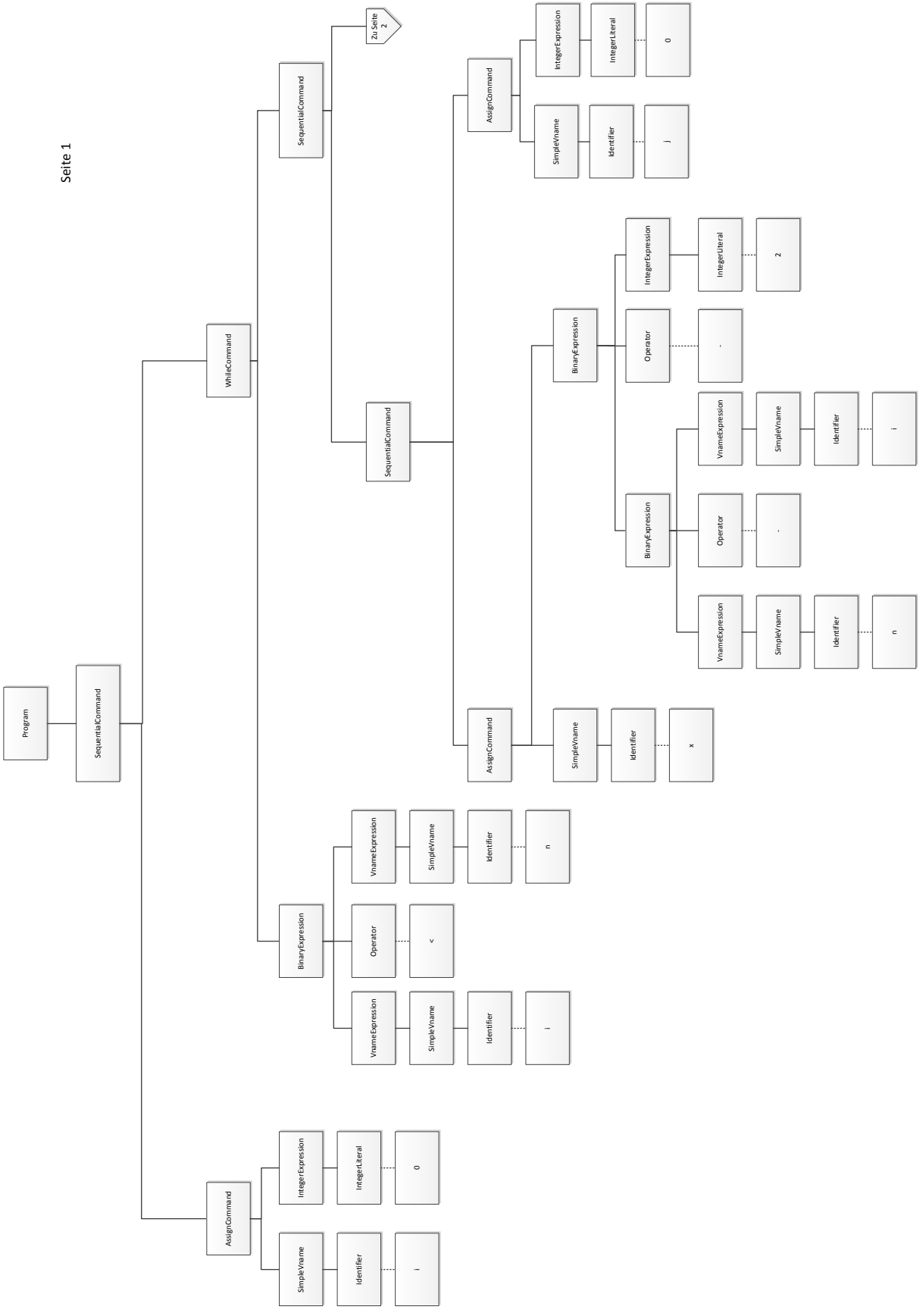
Aufgabe 1.6 Lösung

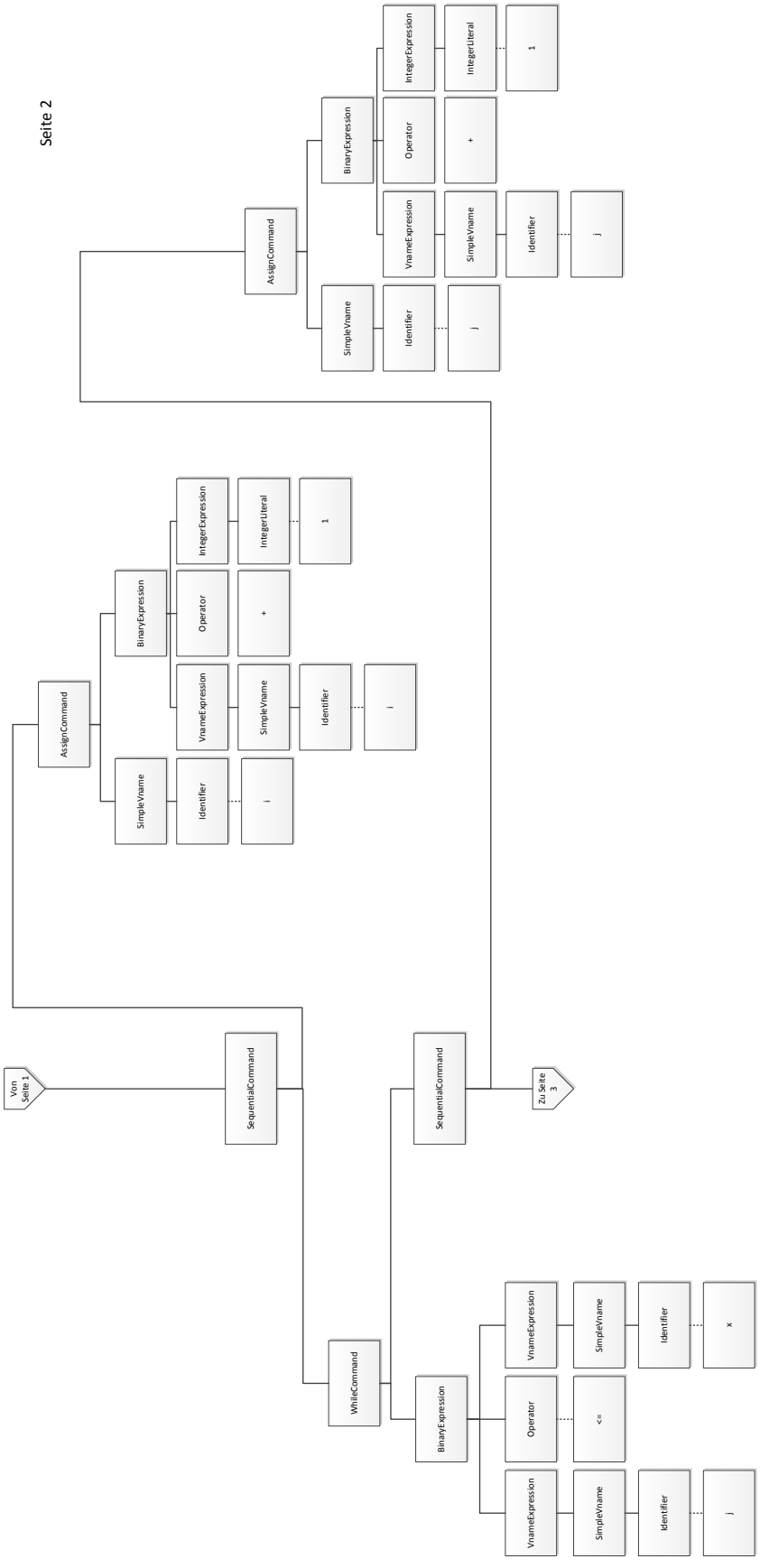
```
private byte scanToken() {
  switch (currentChar) {

  case '0'
    takeIt();
    if(isDigit_8(currentChar)) {
      while(isDigit_8(currentChar))
        takeIt();
      return Token.OctalLiteral
    }
    else
      return Token.IntegerLiteral;

  case '1' ... case '9'
    takeIt();
    while(isDigit(currentChar))
      takeIt();
    return Token.IntegerLiteral;

  ...
  }
}
```

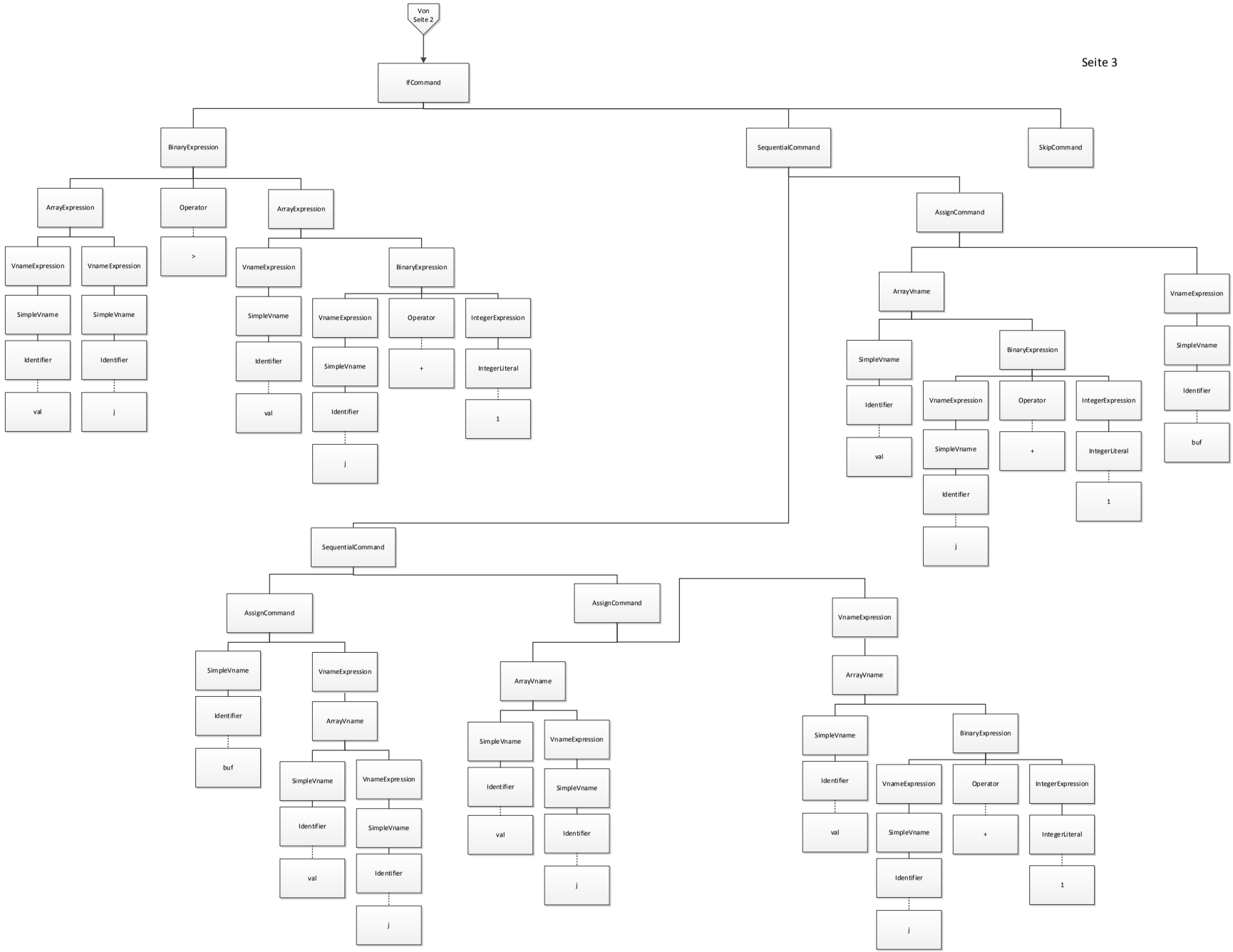
Die gegebene Grammatik ist nicht eindeutig. Wenn im Zeichenstrom beispielsweise das Zeichen *0* vorliegt und mit einem neuen Token begonnen wird, so ist unklar, ob hier ein Integer Literal oder ein Octal Literal beginnt. Dieses Problem kann nur auf kompliziertem Weg mittels Präzedenzregeln o. a. gelöst werden.

Plagiarismus

Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Weitere Infos unter www.informatik.tu-darmstadt.de/plagiarism


Program
SequentialCommand
AssignCommand
SimpleVname
Identifier
i
IntegerExpression
IntegerLiteral
0
WhileCommand
BinaryExpression
VnameExpression
SimpleVname
Identifier
i
Operator
<
VnameExpression
SimpleVname
Identifier
n
SequentialCommand
AssignCommand
SimpleVname
Identifier
x
BinaryExpression
VnameExpression
SimpleVname
Identifier
i
Operator
-
VnameExpression
SimpleVname
Identifier
n
Operator
-
IntegerExpression
IntegerLiteral
2
SequentialCommand
AssignCommand
SimpleVname
Identifier
j
IntegerExpression
IntegerLiteral
0
Zu Seite 2


Seite 2
Von Seite 1
SequentialCommand
WhileCommand
BinaryExpression
VnameExpression
SimpleVname
Identifier
x
Operator
<=
VnameExpression
SimpleVname
Identifier
j
SequentialCommand
Zu Seite 3
AssignCommand
SimpleVname
Identifier
i
BinaryExpression
Operator
+
VnameExpression
SimpleVname
Identifier
i
IntegerExpression
IntegerLiteral
1
AssignCommand
SimpleVname
Identifier
j
BinaryExpression
Operator
+
VnameExpression
SimpleVname
Identifier
j
IntegerExpression
IntegerLiteral
1


Von Seite 2
IfCommand
BinaryExpression
ArrayExpression
VnameExpression
SimpleVname
Identifier
val
VnameExpression
SimpleVname
Identifier
j
Operator
>
ArrayExpression
VnameExpression
SimpleVname
Identifier
val
BinaryExpression
VnameExpression
SimpleVname
Identifier
j
Operator
+
IntegerExpression
IntegerLiteral
1
SequentialCommand
SkipCommand
AssignCommand
ArrayVname
SimpleVname
Identifier
val
BinaryExpression
VnameExpression
SimpleVname
Identifier
j
Operator
+
IntegerExpression
IntegerLiteral
1
VnameExpression
SimpleVname
Identifier
buf
SequentialCommand
AssignCommand
SimpleVname
Identifier
buf
VnameExpression
ArrayVname
SimpleVname
Identifier
val
VnameExpression
SimpleVname
Identifier
j
AssignCommand
ArrayVname
SimpleVname
Identifier
val
VnameExpression
SimpleVname
Identifier
j
VnameExpression
ArrayVname
SimpleVname
Identifier
val
BinaryExpression
VnameExpression
SimpleVname
Identifier
j
Operator
+
IntegerExpression
IntegerLiteral
1