Objektorientierte Programmierung

Kapitel 4: Syntaxdiagramme (und Grammatikregeln)

Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2013/14

http://www.informatik.uni-halle.de/~brass/oop13/

Inhalt

Syntax-Formalismen (1)

- Ein Java-Programm ist eine Folge von Zeichen, aber nicht jede Folge von Zeichen ist ein Java-Programm.
- Ein Alphabet ist eine endliche, nicht-leere Menge, deren Elemente Zeichen heißen.

  Java basiert auf dem Unicode-Zeichensatz. Dies ist eine Obermenge des ASCII-Zeichensatzes, der in Kapitel 1 gezeigt wurde.
- Ein Wort über einem Alphabet ist eine endliche Folge von Zeichen des Alphabets.

  Nach dieser Definition ist ein Java-Programm ein Wort über dem Unicode-Zeichensatz.
- Eine formale Sprache (über einem Alphabet) ist eine (meist unendliche) Menge von Worten (über dem Alphabet).

Syntax-Formalismen (2)

- Es gibt verschiedene Formalismen, um klar und eindeutig eine formale Sprache (Menge von Zeichenfolgen) zu definieren, z.B.
  - Syntax-Diagramme
  - (Kontextfreie) Grammatik-Regeln

    BNF (Backus-Naur-Form) ist eine spezielle Syntax für kontextfreie Grammatikregeln (recht verbreitet).
- Syntaxdiagramme und kontextfreie Grammatiken haben die gleiche Ausdruckskraft, d.h. können die gleichen Mengen von Zeichenfolgen beschreiben.

Syntax-Formalismen (3)

- Als professioneller Programmierer sollte man die Definition der Programmiersprache lesen können.

  Im ersten Semester, wenn Sie mit der Programmierung gerade anfangen, müssen Sie noch nicht die "Java Language Specification" lesen können, sondern nur die Syntaxdigramme auf den Folien dieser Vorlesung.

- Die Grammatik ist eine Referenz in unklaren Fällen, hilft aber auch zum Verständnis der Sprache.

  Die syntaktischen Kategorien sind oft nützliche Konzepte.

- Es könnte ja sein, dass der Compiler einen Fehler enthält.

  Man sollte sich nicht zum "Sklaven" eines einzelnen Compilers machen. Der Compiler hat nicht immer recht (aber meistens schon). Es gibt ein unabhängiges Gesetz (die Java-Spezifikation). Zumindest hilft die Aufklärung eines Fehlers, ihn zukünftig zu vermeiden. Herumprobieren ("wie hätte er es denn gern?") ist alleine keine richtige Lösung.

Syntax-Formalismen (4)

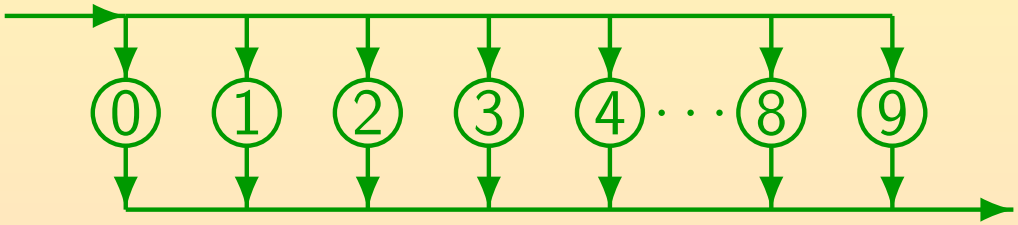
- Mit kontextfreien Grammatiken definiert man nur eine Obermenge der Java-Programme, und stellt dann weitere einschränkende Forderungen, z.B.
  - der Typ des zugewiesenen Wertes muss zum Typ der Variablen passen.
- Kontextfreie Grammatiken (und damit auch Syntaxdiagramme) können solche Bedingungen nicht ausdrücken.

  Dennoch bildet die kontextfreie Grammatik sozusagen das Rückgrat der Sprachspezifikation: Die syntaktischen Kategorien, die in der Grammatik eingeführt sind, werden gebraucht, um die semantischen Zusatzbedingungen überhaupt ausdrücken zu können. Ein allgemeines Prinzip der Programmierung ist, schwierige Aufgaben in Teilaufgaben zu zerlegen. Daher ist man zunächst damit zufrieden, dass die kontextfreie Grammatik eine Obermenge der gültigen Java-Programme beschreibt, in einem zweiten Schritt schränkt man die Menge dann durch Zusatzbedingungen weiter ein.

Inhalt

Syntaxdiagramme (1)

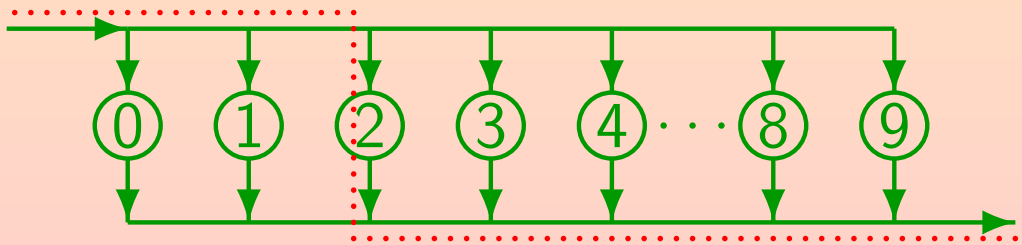
- Digit:



- Ein Syntaxdiagramm besteht aus:
  - Namen der definierten syntaktischen Kategorie,
  - Startknoten: Pfeil von außen in das Diagramm.
  - Zielknoten: Pfeil, der das Diagramm verlässt.
  - Kreise/Ovale und Rechtecke, die mit gerichteten Kanten verbunden sind.

Syntaxdiagramme (2)

- Die formale Sprache, die durch dieses Diagramm definiert wird, ist die Menge der Dezimalziffern $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
- Um zu prüfen, ob eine Eingabe, z.B. “2”, zu der Sprache “Digit” gehört, die durch das Diagramm definiert wird, muss man einen Weg durch das Diagramm finden, der der Eingabe entspricht:



Syntaxdiagramme (3)

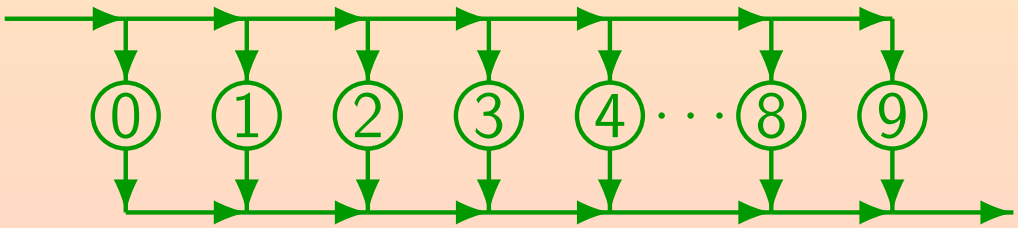
- Solche Diagramme können auf zwei Arten verwendet werden (Synthese und Analyse):
  - Um ein gültiges Wort der Sprache zu erzeugen, folgt man einem Pfad durch das Diagramm vom Start zum Ziel und druckt jedes Symbol in einem Kreis/Oval aus, den man durchläuft.
  - Um festzustellen, ob eine gegebene Eingabe syntaktisch korrekt ist, muss man einen Pfad durch das Diagramm finden, so dass beim Durchlaufen eines Kreises/Ovals das Zeichen darin das nächste Eingabezeichen ist.

Syntaxdiagramme (4)

- Jede Kante hat nur eine mögliche Richtung.

  Manchmal ist es für den Anfänger etwas schwierig, die Richtung einer Kante zu erkennen.

- Wenn alle Richtungen explizit gemacht sind, sieht das Diagramm so aus:



- Normalerweise wird der Pfeilkopf aber nicht in jedem Segment einer Kante wiederholt.

Syntaxdiagramme (5)

- Es gibt Verzweigungsknoten, an denen man verschiedene ausgehende Kanten benutzen kann.

  Z.B., nach der Eingangskante könnte man nach unten gehen, und die Ziffer 0 ausgeben/einlesen, oder man kann nach rechts gehen, um eine der Ziffern 1 bis 9 zu erhalten.

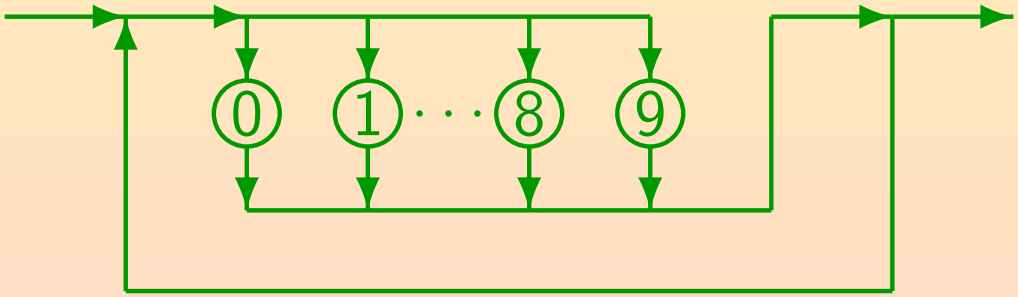
- Um eine gegebene Eingabe zu prüfen, hilft es meist, das nächste Eingabesymbol anzuschauen, um den richtigen Pfad auszuwählen.

  Dies ist, was der Compiler auch macht. Man versucht sicherzustellen, dass an jeder Verzweigung, die Kreise/Ovale, die man in verschiedenen Richtungen erreichen kann, disjunkt sind.

- Gibt es keinen passenden Pfad: Syntaxfehler.

Syntaxdiagramme (6)

- Syntaxdiagramme können Zyklen enthalten (es gibt ja unendlich viele gültige Java-Programme).
- Digit Sequence:



- Aufgabe: Finden Sie einen Pfad durch das Diagramm, um zu zeigen, dass "81" korrekt ist.

  Man darf die gleiche Kante mehrere Male durchlaufen.

Syntaxdiagramme (7)

- Man kann an anderer Stelle definierte Diagramme als Module benutzen.

  Prinzip der Abstraktion: Man kennt schon einfache syntaktische Konstrukte. Diese Konstrukte kann man als elementare Begriffe zur Definition komplexerer syntaktischer Konstrukte benutzen. Entsprechend: Lösung einfacher Aufgaben als elementare Schritte zur Lösung komplexer Aufgaben.

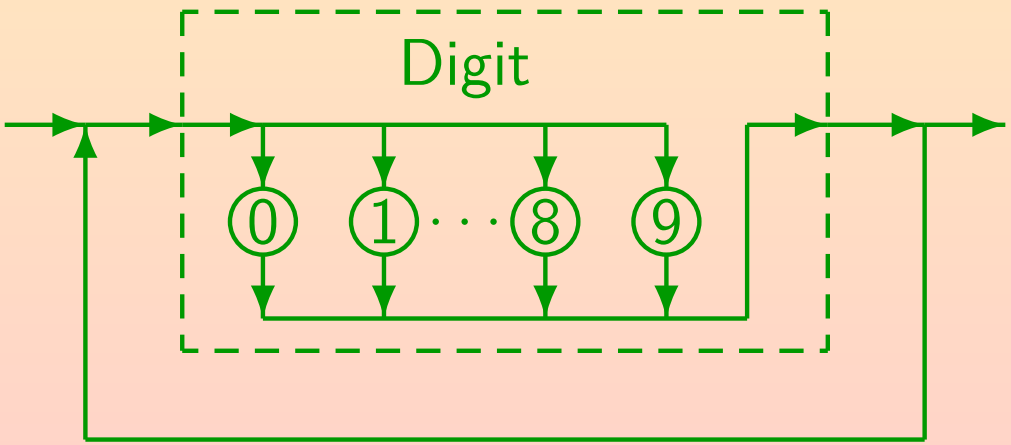
- Dies wird als Rechteck dargestellt, das den Namen des anderen Diagramms enthält:

  Digit Sequence:

- Ein Rechteck steht also für eine syntaktische Kategorie (wie “Subjekt”, “Prädikat”, “Objekt”).

Syntaxdiagramme (8)

- Ein Rechteck kann durch das Diagramm ersetzt werden, für das es steht (es hat wie das Diagramm eine eingehende und eine ausgehende Kante).
- Digit Sequence:



Syntaxdiagramme (9)

- Natürlich muss man nicht explizit das Diagramm der benutzten syntaktischen Kategorie einfügen:
  - Man merkt sich einfach, wo man im übergeordneten Diagramm war (bei welchem Rechteck),
  - durchläuft dann das Diagramm für die syntaktische Kategorie, die in dem Rechteck steht,
  - anschließend (wenn man mit diesem Diagramm fertig ist) kehrt man zum übergeordneten Diagramm zurück, und folgt dort dem Pfeil, der das Rechteck verläßt.

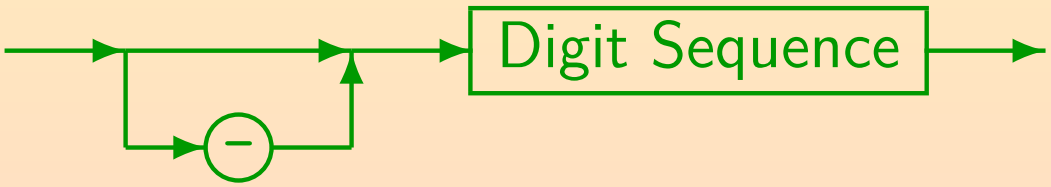
Syntaxdiagramme (10)

- Natürlich weiß man nach einiger Zeit, wofür z.B. die syntaktische Kategorie “Digit” steht, und braucht das zugehörige Diagramm dann nicht mehr nachzuschlagen.
- Jedes Diagramm definiert eine formale Sprache.

  D.h. eine Menge von Zeichenketten.
- Wenn man ein Rechteck durchläuft, kann man jedes Element der zugehörigen Sprache ausdrucken bzw. einlesen.

Syntaxdiagramme (11)

- Kreise/Ovale und Rechtecke können beide im gleichen Diagramm benutzt werden:
- Zahlkonstante: (nur Beispiel, nicht exakt Java)



- Dieses Diagramm definiert eine Sprache, die z.B. folgende Zeichenfolgen enthält: `123`, `-45`, `007`.
- Dagegen gehören folgende Zeichenfolgen nicht zu der definierten Sprache: `+89`, `--5`, `23-42`, `0.56`.

Syntaxdiagramme (12)

- Syntaxdiagramme können "rekursiv" definiert sein, d.h. das Diagramm für eine syntaktische Kategorie X kann selbst einen mit X beschrifteten Kasten enthalten.

  Entsprechend könnten sich auch zwei syntaktische Kategorien X und Y gegenseitig aufrufen. Es ist nicht verlangt, dass eine syntaktische Kategorie vor ihrer Verwendung definiert ist. Es müssen nur am Ende alle verwendeten syntaktischen Kategorien auch wirklich definiert sein.

- Der Pfad, den man für ein Wort der Sprache durch die Diagramme geht, muss immer endlich sein.

  Man kann jetzt zwar nicht mehr vorab die Diagramme für die Kästen vollständig einsetzen, aber man könnte noch die tatsächlich durchlaufenen Kästen nach Bedarf "expandieren".

Inhalt

Grammatik-Regeln (1)

- Man kann die formale Syntax einer Sprache auch mithilfe kontextfreier Grammatik-Regeln definieren.
- Dies geschieht z.B. in der Java-Sprachspezifikation.

  Die Referenz-Handbücher zu den Datenbanksystemen “Oracle” und “DB2” enthalten dagegen Syntax-Diagramme. Beides wird also in der Praxis verwendet. Syntaxdiagramme sind am Anfang intuitiver, aber Grammatik-Regeln sind kompakter.
- Auch in dieser Notation wird unterschieden zwischen
  - syntaktischen Kategorien
    (im Diagramm: Rechteck, jetzt in *kursiver* Schrift) und
  - den letztendlich erzeugten Zeichen der definierten Sprache
    (im Diagramm: Kreis/Oval, jetzt in `Teletype` Schrift).

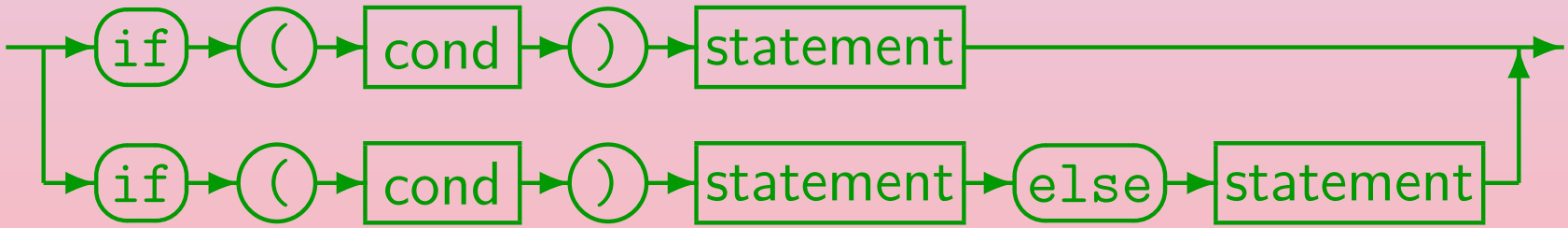
Grammatik-Regeln (2) *

- Beispiel:

  selection-statement:
  `if (` *cond* `)` *statement*
  `if (` *cond* `)` *statement* `else` *statement*

- Dies entspricht folgendem Syntaxdiagramm:

  selection-statement:



- In der obigen Grammatik-Notation ist also
  - zuerst die zu definierende syntaktische Kategorie angegeben,
  - dann eingerückt pro Zeile eine Alternative.

Grammatik-Regeln (3)

- Man wendet Grammatik-Regeln an, indem man ausgehend vom Startsymbol der Grammatik (z.B. *CompilationUnit*)
  - immer eine syntaktische Kategorie durch einen der darunter eingerückten Texte ersetzt,
  - bis der so erzeugte Text schließlich keine syntaktischen Kategorien mehr enthält, sondern nur noch Zeichen der zu definierenden Sprache.
- Weil die Ersetzung bei den Zeichen der erzeugten Sprache (`Teletype`-Schrift) endet, heißen diese auch Terminalsymbole der Grammatik.
- Die syntaktischen Kategorien (Worte in *kursiver* Schrift) heißen entsprechend Nichtterminalsymbole.

Das ganze Wort, d.h. die Kategorie, ist ein Nichtterminalzeichen.

Grammatik-Regeln (4)

- Es gibt viele Notationen für Grammatikregeln:
  - Z.B. kann man syntaktischen Kategorien auch in spitze Klammern $\langle \ldots \rangle$ schreiben.
  - Manche Autoren unterstreichen auch die Zeichen der erzeugten Sprache, oder schreiben sie in '...'.
  - Statt Einrückung und eine Alternative pro Zeile kann man die Alternativen auch durch "|" trennen, und den jeweils letzten Ersetzungstext durch ";" beenden.
  - In der Theorie ist jede Alternative eine eigene Grammatikregel, mit der zu ersetzenden syntaktischen Kategorie (ein Nichtterminalzeichen) auf der linken Seite, und dem Ersetzungstext (Folge von Terminal- und Nichtterminalzeichen) auf der rechten Seite, dazwischen meist "$\rightarrow$".

Grammatik-Regeln (5)

- Im Java-Standard werden noch zwei Abkürzungen benutzt:
  - Wenn die alternativen Ersetzungstexte sehr kurz sind, z.B. einzelne Zeichen, wird nach der zu definierenden syntaktischen Kategorie "one of" geschrieben, und die Alternativen darunter in einer Zeile.

    NonZeroDigit: one of
    `1 2 3 4 5 6 7 8 9`
  - Im Ersetzungstext können syntaktische Kategorien mit dem Index "*opt*" markiert werden: Dann kann man sie an dieser Stelle auch durch die leere Zeichenfolge ersetzen.

    Falls mehrere Zeichen mit "*opt*" markiert sind, ist jedes für sich optional, d.h. man kann eine beliebige Teilmenge weglassen.

    DecimalIntegerLiteral:
    DecimalNumeral $\mathit{IntegerTypeSuffix}_{\mathit{opt}}$

Beispiel: Zahlkonstanten (1)

- Zahlkonstanten in Dezimalschreibweise (z.B. 123) sind in der Java-Spezifikation durch folgende Grammatikregeln definiert:

DecimalNumeral:
0
NonZeroDigit Digits$_{opt}$

Digits:
Digit
Digits Digit

Digit:
0
NonZeroDigit

NonZeroDigit: one of
1 2 3 4 5 6 7 8 9

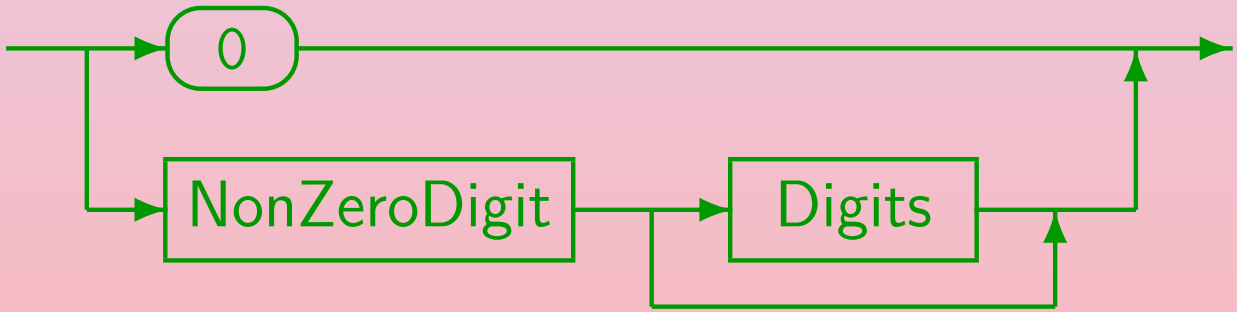
Beispiel: Zahlkonstanten (2)

- Die Regel

DecimalNumeral:
0
NonZeroDigit Digits$_{opt}$

entspricht folgendem Syntaxgraphen:

DecimalNumeral:



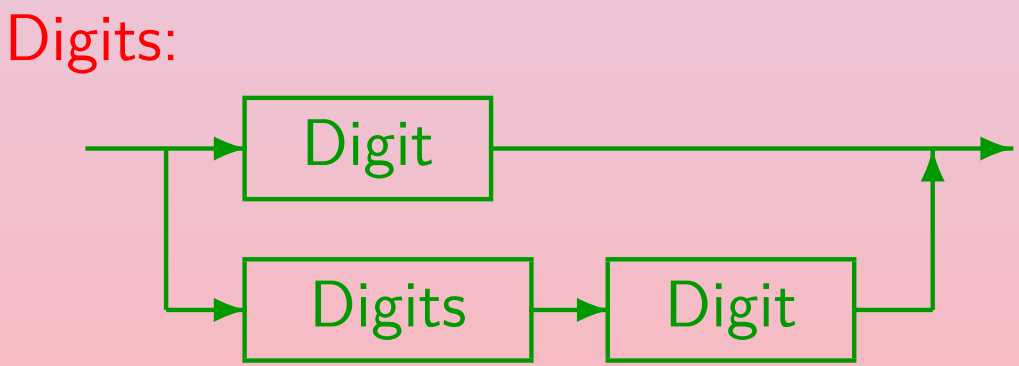
Diese etwas komplizierte Lösung ist gewählt, um führende Nullen auszuschließen, z.B. wäre 009 nicht gültig. Führende Nullen markieren die Oktalschreibweise, siehe nächstes Kapitel.

Beispiel: Zahlkonstanten (3)

- Die Regel

Digits:
Digit
Digits Digit

entspricht folgendem Syntaxgraphen:

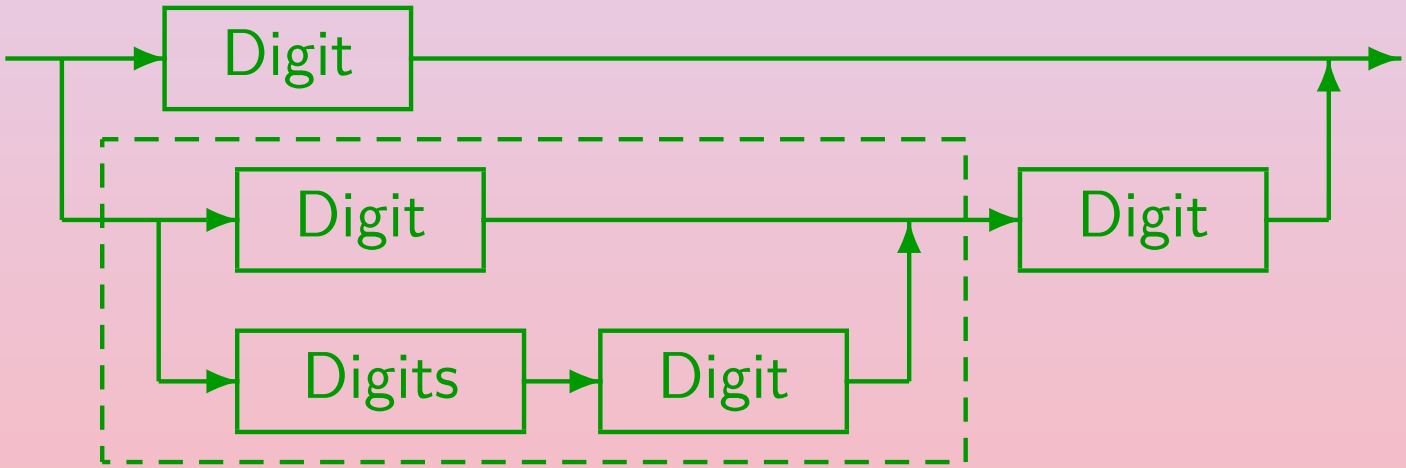


- Diese Definition ist rekursiv: Die syntaktische Kategorie "*Digits*" wird in ihrer eigenen Definition benutzt.

Beispiel: Zahlkonstanten (4)

- Rekursives Diagramm nach einmaliger Einsetzung:

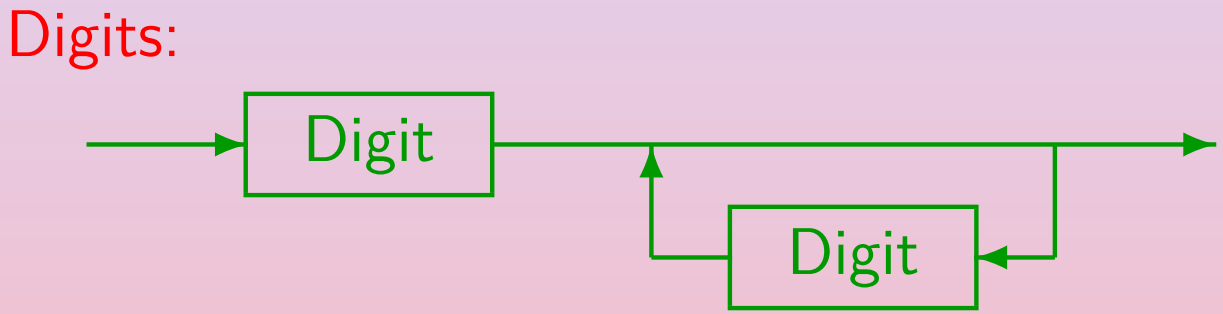
Digits:



- Jedesmal, wenn man den Zweig mit dem rekursiven Aufruf benutzt, erhält man eine Ziffer mehr.

Beispiel: Zahlkonstanten (5)

- In Syntaxdiagrammen sind Rekursionen relativ selten, man würde *Digits* mit einem Zyklus definieren:



Dieses Diagramm ist äquivalent, d.h. definiert die gleiche Sprache.

- In Grammatik-Regeln kann man dagegen keine Zyklen schreiben, und setzt intensiv Rekursion ein.

  Zum Teil hat die dadurch gegebene Struktur durchaus Vorteile, z.B. kann man mit Grammatik-Regeln die implizite Klammerung bei Wertausdrücken präzise definieren, während Zyklen mindestens eine Zusatzangabe zur Assoziativität der Operatoren erfordern (siehe Kapitel 7).

Beispiel: Zahlkonstanten (6)

- Herleitung von 123 durch sukzessive Anwendung der Regeln:

$$
\begin{aligned}
& \textit{DecimalNumeral} \\
\rightarrow\ & \boxed{\textit{NonZeroDigit}\ \textit{Digits}_{opt}} \\
\rightarrow\ & \boxed{1}\ \textit{Digits}_{opt} \\
\rightarrow\ & 1\ \boxed{\textit{Digits}} \\
\rightarrow\ & 1\ \boxed{\textit{Digits}\ \textit{Digit}} \\
\rightarrow\ & 1\ \boxed{\textit{Digit}}\ \textit{Digit} \\
\rightarrow\ & 1\ \boxed{\textit{NonZeroDigit}}\ \textit{Digit} \\
\rightarrow\ & 1\ \boxed{2}\ \textit{Digit} \\
\rightarrow\ & 1\ 2\ \boxed{\textit{NonZeroDigit}} \\
\rightarrow\ & 1\ 2\ \boxed{3}
\end{aligned}
$$

Rot: Nichtterminalzeichen, das ersetzt wird. Kasten in nächster Zeile: Ersetzung.

Hinweis zur Java-Spezifikation *

- Die Java-Spezifikation enthält im Anhang eine weitere Grammatik für die Sprache Java.
- Dies ist nicht einfach eine Zusammenstellung der im Hauptteil angegebenen Grammatik-Regeln, sondern eine Variante, die besser für die Programmierung der Syntaxanalyse geeignet sein soll.

  Die Regeln im Hauptteil sollen dagegen für die Erklärung verständlicher sein.
- Im Anhang wird eine etwas andere Grammatik-Syntax benutzt, in der mit
  - `[...]` optionale Teile,
  - `{...}` wiederholbare Teile, und mit
  - `|` Alternativen markiert sind.