Einführung in die Programmiertechnik

Formale Beschreibung von Programmiersprachen

Lexikalische Regeln

- Definition von „Wörtern“ (Lexem, Token)
  - Gruppierung von Zeichen
- Lexikalische Kategorien: Klassen „ähnlicher“ Wörter
  - Schlüsselwörter (*keywords*): Feste Menge reservierter Buchstabenfolgen (z.B. if, while, else)
  - Bezeichner: Selbstgewählte Namen für Variablen, Funktionen, etc.
    - Python: 1. Zeichen ist Buchstabe oder Unterstrich, gefolgt von beliebig vielen Buchstaben, Ziffern, oder Unterstrich
      - Buchstabe: a..z, A..Z; Ziffer: 0..9
  - Einfache oder zusammengesetzte Spezialsymbole:
    - + - * / = < > <= >= ; .
  - Literale: Zahlenkonstanten, Zeichen(ketten)konstanten
- Lexikalische Mehrdeutigkeit: Wie weit erstreckt sich ein Lexem?
  - "Hallo"+"Welt"

Schlüsselwörter in Python

and	del raise	for	is	
assert	elif	from	lambda	return
break	else	global	not	try
class	except	if	or	while
continue	exec	import	pass	yield
def	finally	in	print	

Lexikalische Regeln (2)

- Trenner zwischen Lexemen: Freiraum und Kommentare
- Python: Unterscheidung zwischen Leerzeichen, Tabulator, Zeilenumbruch
  - Tabulator rückt bis zur nächsten durch 8 teilbaren Spalte ein
- Kommentare in Python: Beginnend mit #, bis Zeilenende

  # Innerhalb des Kommentars ist beliebiger Text erlaubt.
- Kein Leerzeichen ist erforderlich, wenn auf einen Bezeichner ein Sonderzeichen folgt
  - `return sin(3.0)` # Leerzeichen zwischen return, sin
    - keins zwischen sin, (, zwischen (, 3.0, zwischen 3.0,)
- Relevanz von Groß- und Kleinschreibung:
  - C, Java, Python: Unterschied ist relevant, Schlüsselwörter klein
  - Pascal, SDL-92: keine Unterscheidung
  - Modula-2: Unterschied ist relevant, Schlüsselwörter groß

Lexikalische Regeln (3)

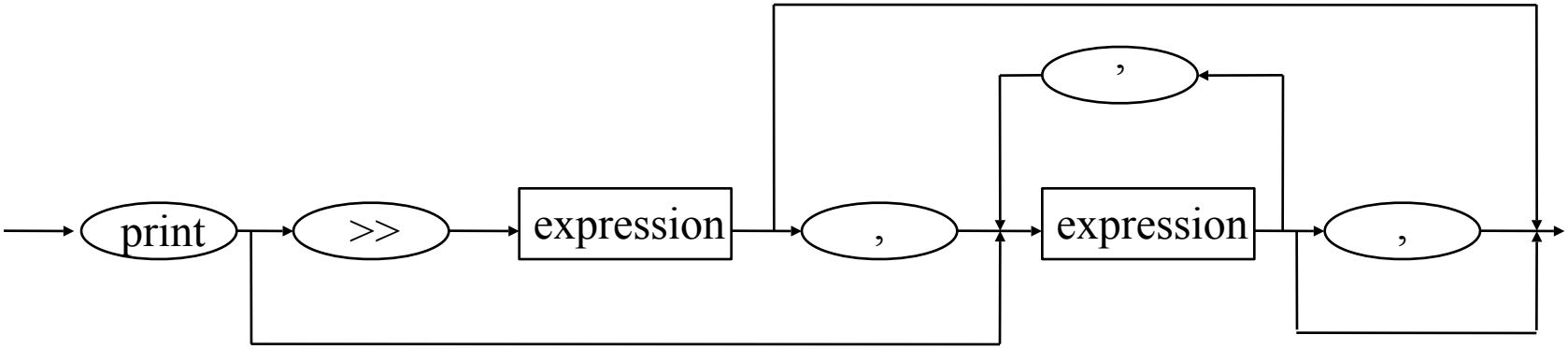
- Stil-Regeln (coding conventions): Verwendung von Freiraum, Formatierung von Kommentaren, ...
  - üblich: zusammengehörige Anweisungen werden gleich weit eingerückt (Python: keine Konvention, sondern durch Syntax verlangt)
    - C, Java: wo werden die öffnenden und schließenden Klammern geschrieben?
  - üblich: nach Kommas (z.B. in Parameterlisten) wird ein Leerzeichen gesetzt
    - Leerzeichen auch vor und nach letztem Argument einer Funktion, auch vor öffnender Klammer?
    - `anmelden(vorname, nachname, matrikel)`
    - `anmelden( vorname, nachname, matrikel )`
    - `anmelden (vorname, nachname, matrikel)`

Syntaktische Regeln

- Allgemein: Sprache ist Menge von Symbolfolgen
  - Alphabet A, Sprache $L \subset A^*$
  - Alphabet: i.d.R. abstrakte Zeichen (Lexeme)
- Hilfssymbole (*non-terminals*) und "Zeichen" des Alphabets (Terminalsymbole)
- Definition der Sprache mithilfe einer Grammatik
  - verschiedene Grammatikkalküle, z.B. reguläre Grammatik, kontextfreie Grammatik
- Kontextfreie Grammatik: Hilfssymbole werden unabhängig von ihrem Auftreten im Gesamtprogramm definiert
  - Definition über Syntaxdiagramm oder EBNF

Syntaxdiagramm

- gerichteter Graph mit genau einem Eingang und genau einem Ausgang
  - verschiedene Notationen, z.B. Rechtecke für Nonterminals und Ellipsen für Terminals
- Beispiel: print-Anweisung in Python



EBNF-Regel

- Verschiedene Notationen
- Hilfssymbole werden mit ::=-Regeln definiert
- Komposition von Regeln:
  - Hintereinanderschreiben von Hilfssymbolen und Terminalsymbolen
  - Alternative Regeln für ein Hilfssymbol: R1 | R2
  - Optionale Folge von Symbole: [S1 S2 ...]
  - beliebige Wiederholung (auch 0-mal): (S1 S2 ...) *
  - beliebige Wiederholung (wenigsten 1-mal): (S1 S2 ...)+
- Beispiel: print-Anweisung in Python

```
print_stmt  ::=    "print" ( [expression ("," expression)* [","]]
                   | ">>" expression [("," expression)+ [","]] )
```

Mehrdeutigkeit von Grammatik

- u.U. mehrere Möglichkeiten, eine Anweisung mit der Grammatik in Übereinstimmung zu bringen (zu *parsen*)
- Beispiel: if-Anweisung in C, Java, ...

  if_stmt ::= “if“ "("condition “)“ statement [“else“ statement]

  statement ::= simple_stmt | while_stmt | if_stmt | ...

  – Was bedeutet dann

  if (x >= y) then if (x > y) then a=x; else b=y; else c=z;
- Auflösung der Mehrdeutigkeit: z.B. durch zusätzliche (natürlichsprachige) Regeln
  – Beispiel: „Der else-Zweig gehört stets zum letztmöglichen if-Teil“

(Statische) Semantische Regeln

- Nicht alle Sätze der Sprache L sind „sinnvoll“
  - Programmiersprache ist nicht kontextfrei
  - Versuche, alle Regeln für korrekte Programme in Grammatik zu erfassen, sind in der Vergangenheit meist gescheitert (Ausnahme: z.B. Algol-68)
- Zusätzlich zur Grammatikdefinition weitere Regeln
  - Einhaltung aller Regeln: Wohlgeformtheit (well-formedness)
    - Verletzung von Wohlgeformtheit: ill-formed (fälschlich oft: illegal)
- z.B. C, C++, Java: Variablen müssen vor Verwendung deklariert sein
  - Java, Python: lokale Variablen müssen vor Verwendung einen Wert erhalten haben
- Regeln oft in natürlicher Sprache formuliert
  - u.U. auch mit mathematischen Formalismen, z.B. Prädikatenkalkül

(Dynamische) Semantik von Programmen

- Ist ein Programm wohlgeformt: was „bedeutet“ das Programm?
  - Antwort durch dynamische Semantik
- Funktionale Sicht: Programm ist eine mathematische Funktion; Semantik beschreibt Parameter und Ergebnis
  - Begriff der berechenbaren Funktion
- Operationale Sicht: Programm ist Folge von Aktionen
  - Trace-Semantik: Bedeutung eines Programms ist die Menge der möglichen Traces
  - Begriff der beobachtbaren Aktion: Ein/Ausgabe, Schreiben auf „wichtige“ Variablen (C: Variablen, die als `volatile` deklariert sind)
    - as-if rule: zwei Programme sind gleich, wenn sie bei gleicher Eingabe die gleiche Folge beobachtbarer Aktionen erzeugen
- Gleichheit von Programmen im Allgemeinen nicht entscheidbar (Halteproblem)