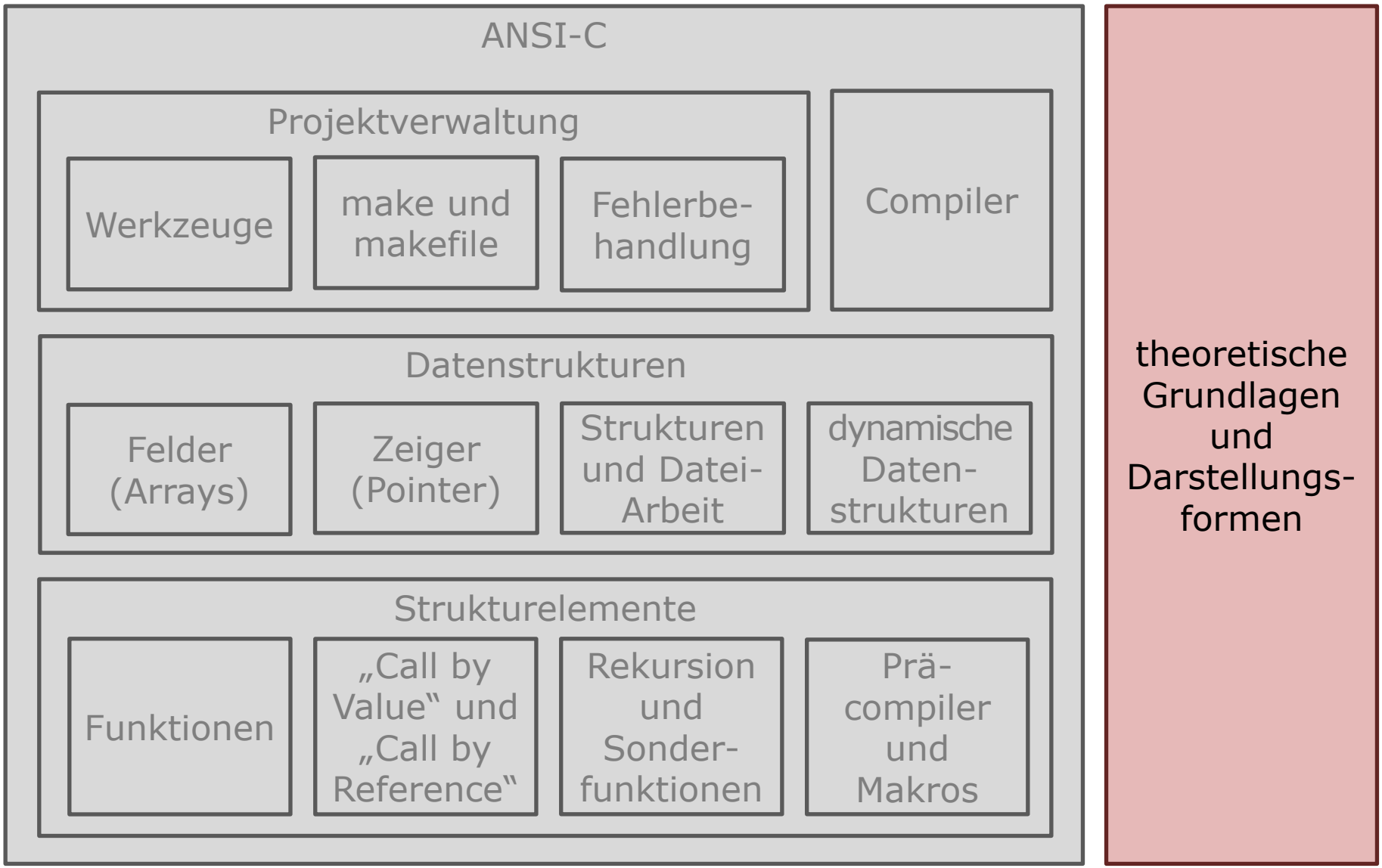
BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

Imperative Programmierung

Theoretische Grundlagen und Darstellungsformen

Prof. Dr.-Ing. Tenshi Hara
tenshi.hara@ba-sachsen.de

Aufbau der Lehrveranstaltung



Informatik

Informatik ist die Wissenschaft von der systematischen Verarbeitung von Daten, insbesondere der automatischen Verarbeitung mit Hilfe von Digitalrechnern.

(Duden Informatik)

- Informatik ist
  - weder eine Natur- noch eine Technikwissenschaft
  - eher eine aus der Mathematik hergeleitete Wissenschaft
    - Untersuchung und Behandlung vom Menschen geschaffener formaler Strukturen (Datenstrukturen, Sprachstrukturen, Systemstrukturen) → Disziplin der Informationswissenschaften
    - Lösung von formalisierten Problemen → Ingenieurwissenschaft
- zentrales Thema der Informatik: Formulierung und Realisierung von Algorithmen zur Lösung von Problemen

Algorithmus

Ein Algorithmus ist eine (eindeutige) Handlungsvorschrift zur Lösung eines Problems. Er besteht aus endlich vielen (wohldefinierten) Einzelschritten.

- Algorithmen müssen von einer Maschine ausführbar sein
  - Notation unter Einhaltung formaler Regeln (Syntax)
  - Bedeutung der Konstrukte (Semantik) muss nicht bekannt sein
  - Nutzung von formalen Sprachen (Programmiersprachen)
- in einer Programmiersprache formulierter Algorithmus ist ein Programm
- Programmierung: alle Aktivitäten von der Problemstellung bis zum fertigen Programm

Generationen von Programmiersprachen (1/3)

- erste Generation: Maschinensprachen
  - Befehle und Adressen werden in Binärform eingegeben
  - nur auf dem jeweiligen Prozessortyp lauffähig (keine Interpretation)
  - in der Anfangszeit der Computer vor allem als physisch gesteckte oder gelochte Befehle
- zweite Generation: Assembler-Sprachen
  - für die Maschinenbefehle und -befehlssequenzen werden für Menschen leichter merkbare Bezeichner (Mnemonic) eingeführt
  - für Adressen werden symbolische Namen verwendet
  - nur auf der jeweiligen Computer-Architektur lauffähig
  - für einen Prozessortyp kann es mehrere Assembler-Sprachen geben
  - Notwendigkeit von Übersetzungsprogrammen (Assembler)

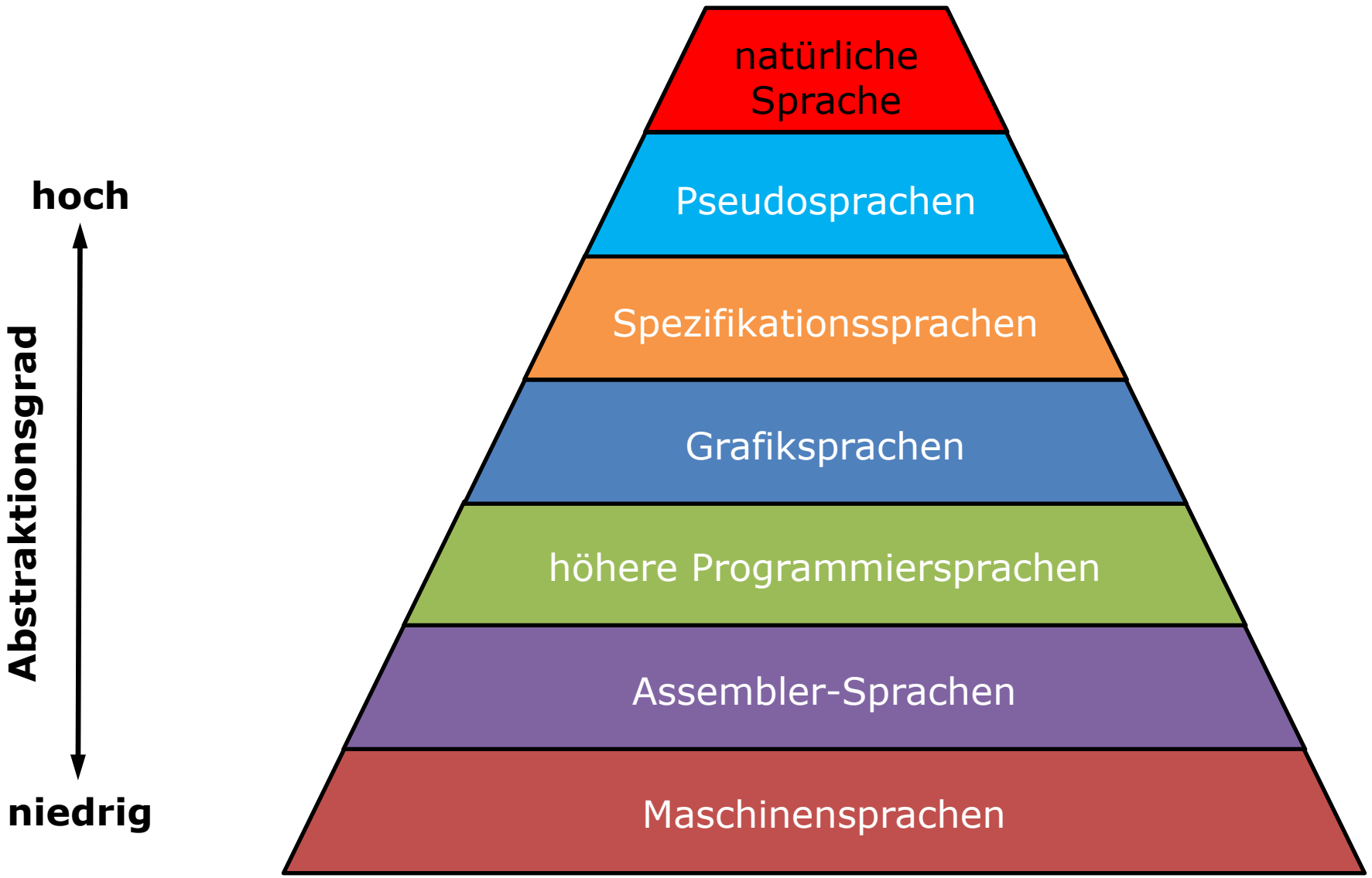
Generationen von Programmiersprachen (2/3)

- dritte Generation: problemorientierte Sprachen
  - höhere Programmiersprachen; oft auch als prozedurale Sprachen bezeichnet (Beschreibung der Algorithmen und Abläufe)
  - Einführung organisatorischer Sprachelemente: Folge (Sequenz), Verzweigung (Selektion) und Schleife (Iteration)
  - Möglichkeiten für strukturierte Programmierung
  - Quellprogramme werden durch Compiler in Zwischenkodedateien (Objekt-Dateien), architekturspezifischen Assembler-Kode oder direkt in prozessorspezifischen Maschinenkode übersetzt
  - Objekt-Dateien werden durch Linker in Maschinenkode übersetzt
  - Beispiele: FORTRAN, COBOL

Generationen von Programmiersprachen (3/3)

- vierte Generation: deklarative Sprachen / nichtprozedurale Sprachen
  - beschreiben, was das Programm leisten soll
  - kein Einfluss auf interne Abläufe bei der Umsetzung
  - meist auf ganz bestimmte Aufgabengebiete zugeschnitten (Datenbanken, Tabellenkalkulation, …)
  - Beispiele: SQL, Open Access
- fünfte Generation: logische Sprachen
  - beschreiben anhand von Logik-Regeln, was das Programm leisten soll
  - können verwendet werden, regelbasierte Intelligenz oder allgemeine Intelligenz nachzuvollziehen (erster Schritt Richtung KI)
  - spezielle Aufgabengebiete wie Spracherkennung, Wissensverarbeitung und Robotik
  - Beispiele: Lisp, Prolog, Smalltalk

Darstellungsformen – Aufwandspyramide



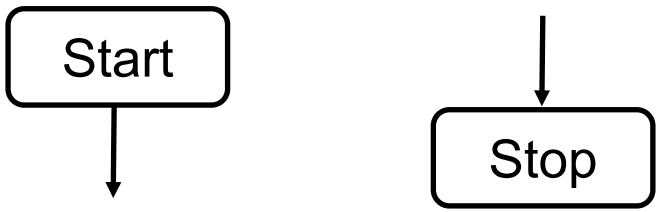
Darstellungsformen – natürliche Sprache

- verbale Beschreibung der einzelnen Schritte
  - oft schwer verständlich (Übersetzungsprobleme, Bedeutung von Wörtern, Zweideutigkeiten, ...)
  - Beschreibung muss sehr genau sein und kann deshalb lang und unübersichtlich werden
- Beispiel**:**

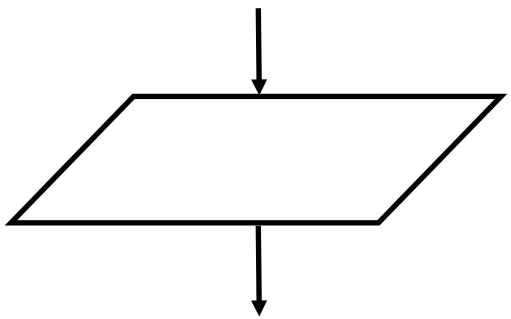
  Größter gemeinsamer Teiler zweier natürlicher Zahlen
  (nach Euklid von Alexandria ca. 365–300 v. Chr.)
  „Nimmt man abwechselnd immer das Kleinere vom Größeren weg, dann muss der Rest schließlich die vorhergehende Größe messen …“
  (Die Elemente, Zehntes Buch § 3)

Darstellungsformen – PAP (DIN 66 001) (1/2)

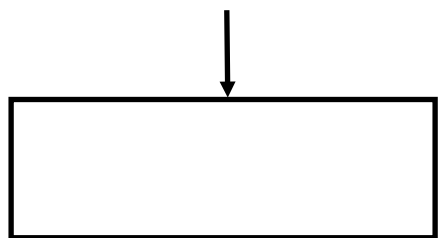
- Anfang / Ende

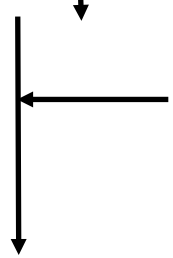


- Ein- / Ausgabe



- Anweisung (Aktion)



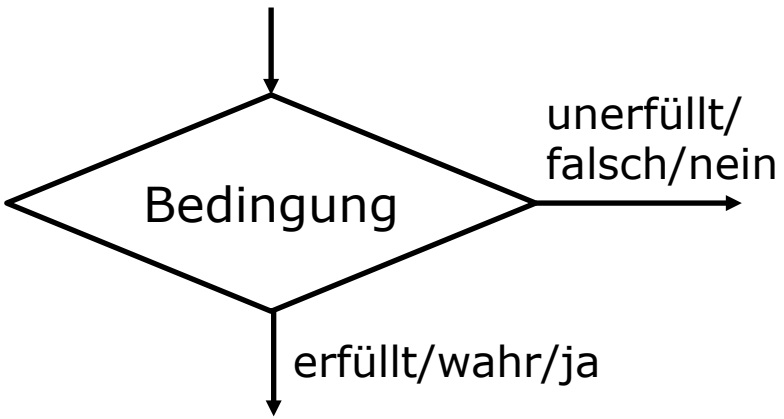


- Zusammenführung

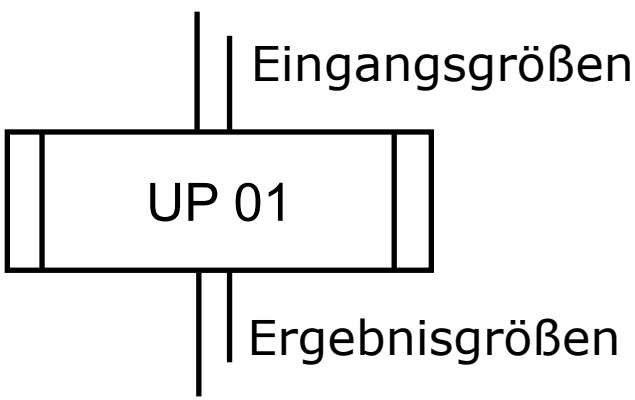
Darstellungsformen – PAP (DIN 66 001) (2/2)

- Anschluss-Stellen

- Verzweigung



- Aufruf von Teilalgorithmen



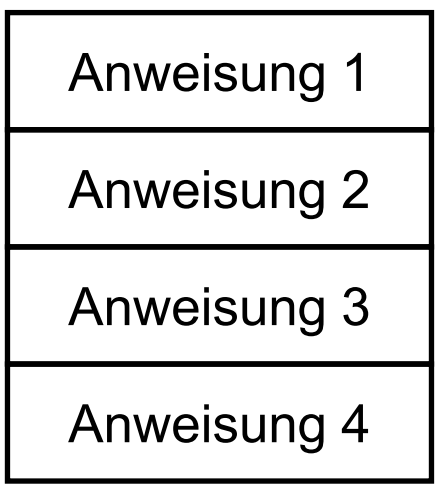
Darstellungsformen – Struktogramm (1/4)

Nassi-Shneiderman-Diagramm (DIN 66 261)

- Grundform ist ein Strukturblock (Rechteck)
- Abarbeitung erfolgt „von oben nach unten“

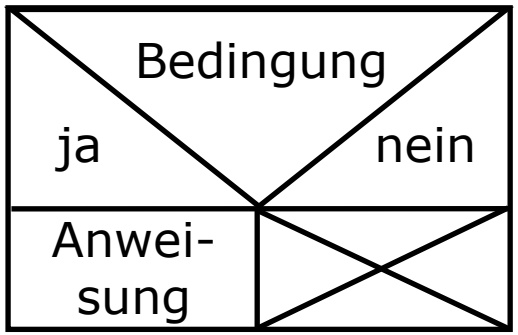
- Anweisung

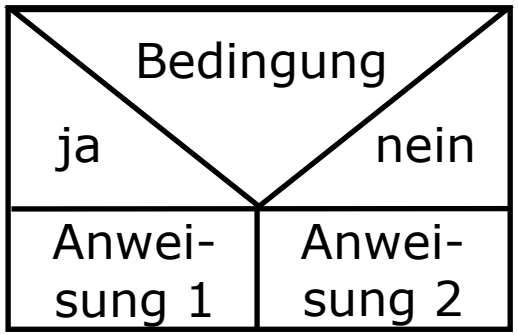
- Sequenz/Folge



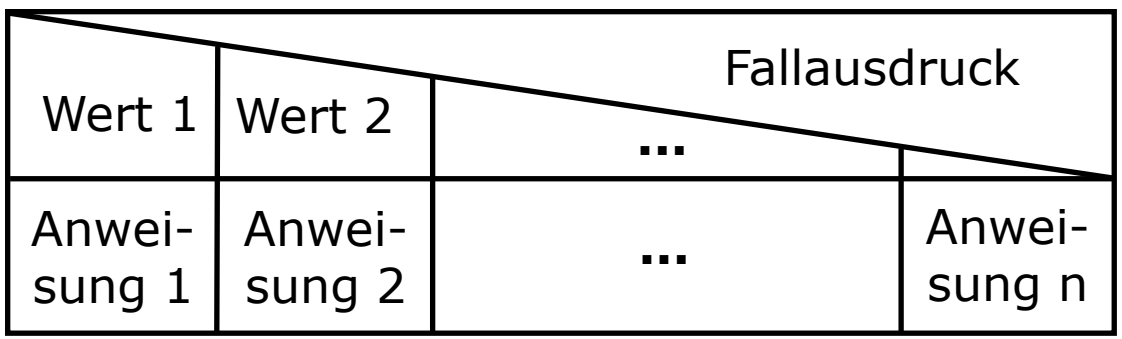
Darstellungsformen – Struktogramm (2/4)

- Selektion/Verzweigung mit/ohne Alternative (if bzw. if|else)

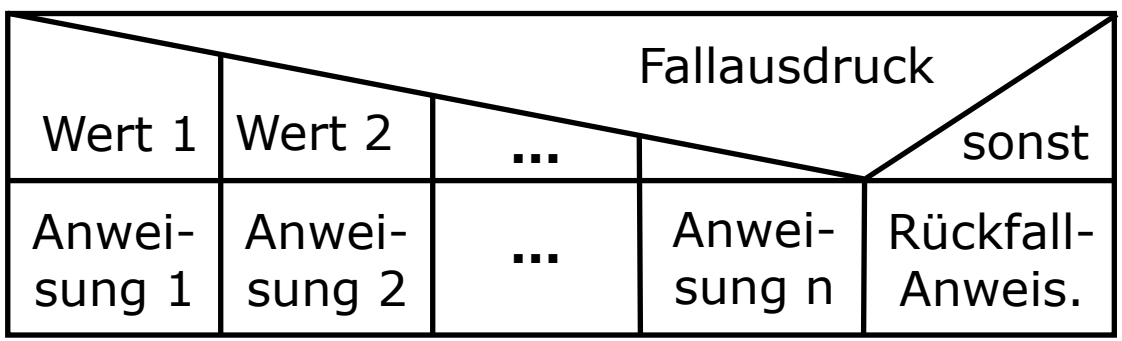




- Mehrfachverzweigung *ohne* Rückfall (switch)

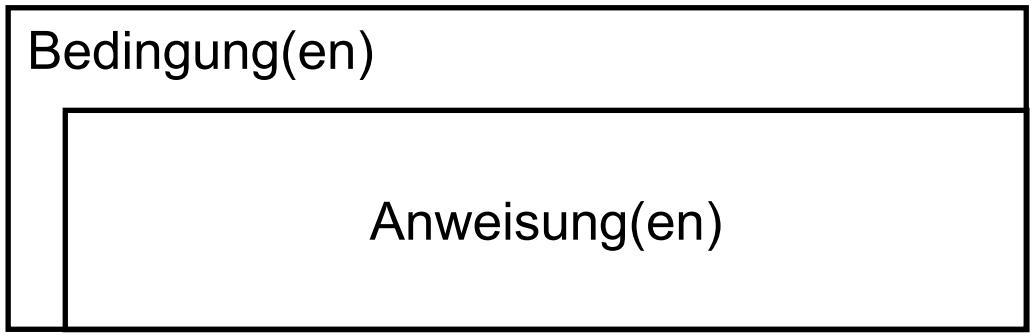


- Mehrfachverzweigung *mit* Rückfall (switch|default)

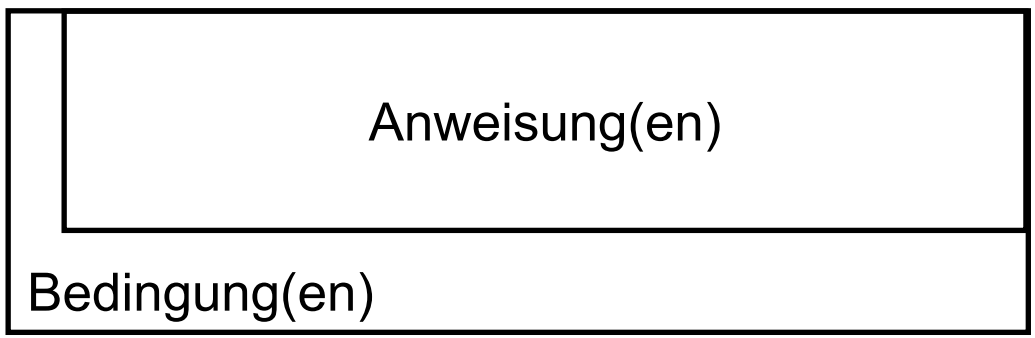


Darstellungsformen – Struktogramm (3/4)

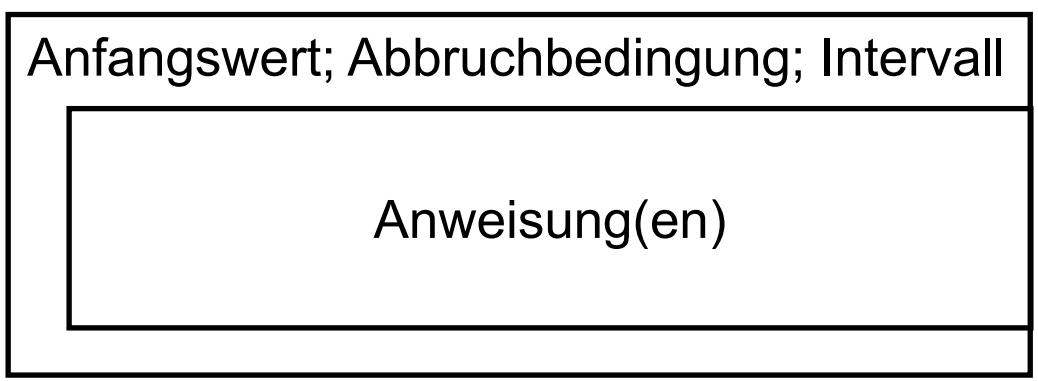
- kopfgesteuerte Schleife („Solange Bedingung erfüllt ist, wiederhole das Folgende“) (abweisender Zyklus)



- fußgesteuerte Schleife („Wiederhole das Folgende bis Bedingung nicht mehr erfüllt ist“) (mindestens 1 Durchlauf)

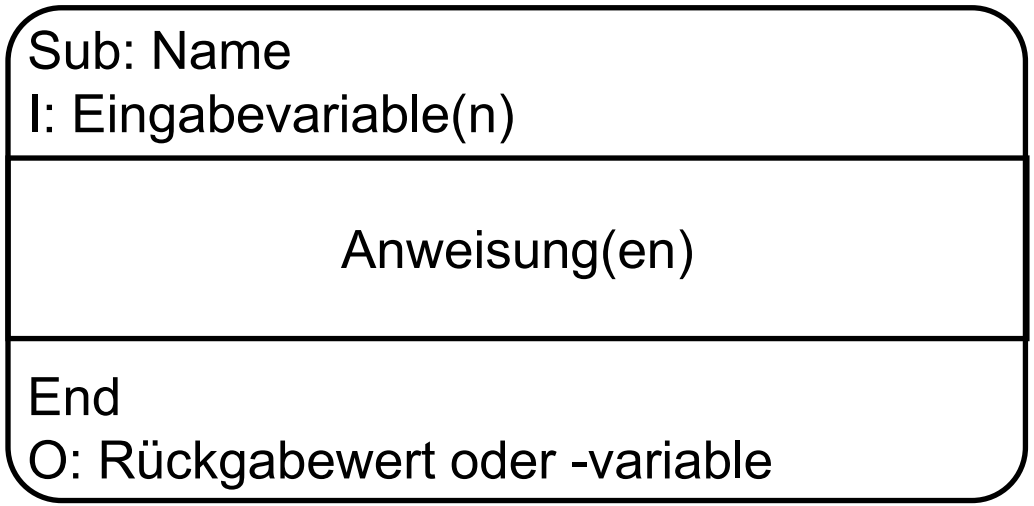


- Schleife von Anfangswert bis Endwert (Zählschleife)



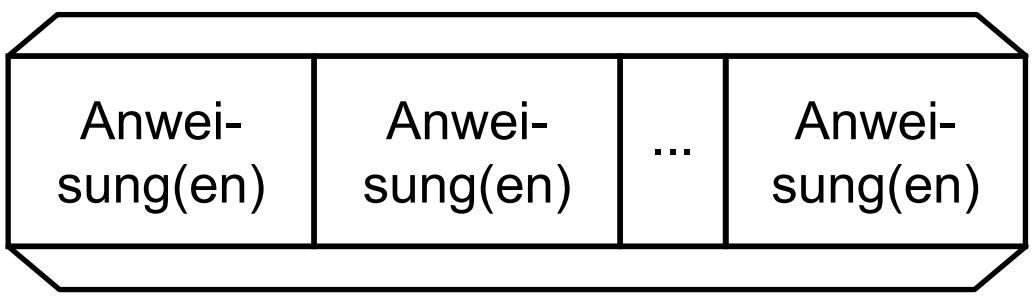
Darstellungsformen – Struktogramm (4/4)

- Unterprogramm
  (kann nach Definition anhand des Namens wie normale Anweisung verwendet werden)



- Abbruchelement
  - bricht Verzweigungen, Mehrfachverzeigungen, Schleifen und Unterprogramme ab
  - Programm setzt direkt hinter dem das Abbruchelement enthaltenden Element fort

- parallele Ausführung von Blöcken



Darstellungsformen – Pseudo-Sprachen

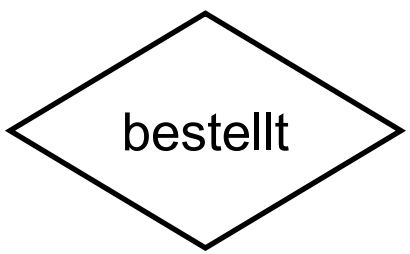
- dienen oft zur sprachunabhängigen Beschreibung von Algorithmen
- Schlüsselwörter lehnen sehr stark an existierende Programmiersprachen
- Beispiel: Heron-Verfahren

```
Heron(Eingabe: a) beginne:
   Hilfsvariablen: x, y;
   x := a; y := 1;
   Wiederhole solange ((x*x - a) > 0.000001) {
      x := (x+y)/2;
      y := a/x;
   }
   Gib als Ergebnis zurück: x
Ende
```

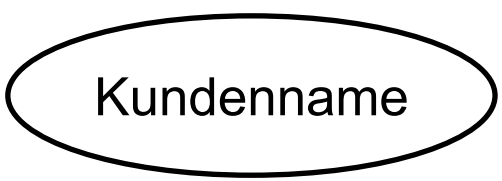
Darstellungsformen – ERM (Entity-Relationship-Modell)

- Orientierung an Daten und deren Beziehungen
- Entity-Relationship-Modell (ERM) ist ein Datenmodell
- ERM dient nicht zur Beschreibung von Algorithmen
  - Datenobjekt (Entity)

  - Beziehung (Relationship)



  - Eigenschaft eines Objekts (Attribute)



Programmiersprachen – Vorbetrachtungen

- theoretisch könnten alle Programme mit einer einzigen Programmiersprache programmiert werden
- es sind nur Sequenzen, Selektionen und Schleifen/Zyklen notwendig
- verschiedene an die Aufgabenstellung angepasste Programmiersprachen für verschiedene Aufgabenstellungen (Wirtschaft, mathematische Verfahren, künstliche Intelligenz, …)
- komplexer werdende Programme und Programmsysteme bringen durch neue Betrachtungsweisen neue Programmiersprachen dafür hervor
- Klassifizierung von Programmiersprachen erfolgt anhand Sprachtyps

Programmiersprachen – Klassifizierung (1/4)

- imperative Programmiersprachen
  - anweisungsorientierte Programmiersprachen
  - abstrakte Datentypen und Modularisierung
  - Erstellen von Programmen für von-Neumann-Rechner
- prozedurale Programmiersprachen
  - gehören zu den imperativen Programmiersprachen
  - Sprachen der ersten bis dritten Generation
  - Zergliederung der Aufgaben in Unterprogramme
  - Gebrauch von Variablen, Befehlen, Prozeduren (geben keinen Wert zurück) und Funktionen (können Wert zurück geben)
  - Nutzung von Modulen und Bibliotheken
  - C-Beispiel: `int sum (int a, int b) { return(a + b); }`

Programmiersprachen – Klassifizierung (2/4)

- deklarative (deskriptive) Programmiersprachen
  - Beschreibung des Problems durch funktionale (Funktionen)
  - oder durch logische Zusammenhänge (Programmklauseln)
- logische Programmiersprachen
  - Lösung von bestimmten Aufgaben (bspw. Datenbanken)
  - es werden nur die Bedingungen für eine korrekte Lösung bestimmt (es wird kein Lösungsweg angegeben)
  - die Einzelheiten der Umsetzung werden nicht betrachtet
  - auch prädikative Programmiersprache (basierend auf dem Prädikatenkalkül); Rekursion und Backtracking
  - PROLOG-Beispiel: `grossvater(X,Z) :- vater(X,Y), vater(Y,Z)`

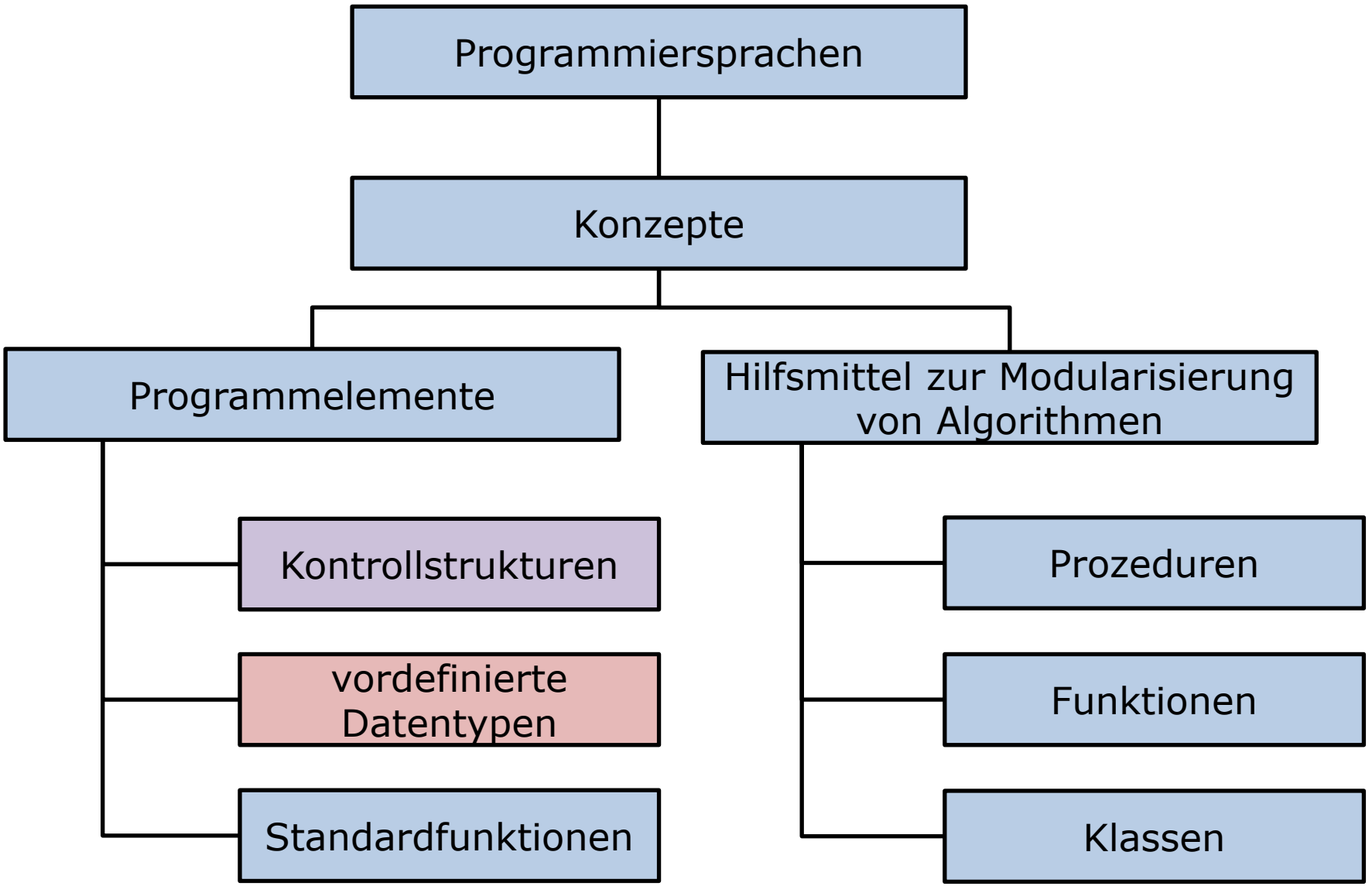
Programmiersprachen – Klassifizierung (3/4)

- funktionale Programmiersprachen
  - Charakterisierung:
    1. große Anzahl eingebauter Funktionen
    2. kompakte Funktionsdefinition
    3. rekursive Aufrufbarkeit von Funktionen
    4. bedingte Ausdrücke
    5. Programme sind Terme
    6. Formeln werden als besondere Terme aufgefasst
    7. keine Einschränkung in Zahl und Typ der Argumente und Funktionswerte bei selbstdefinierten Funktionen
    8. Argumente und Wert von Funktionen dürfen Funktionen sein
  - basierend auf dem λ-Kalkül
  - idealerweise ohne Variablen
  - LISP-Beispiel: `(DEFINE (SUM A B) (+ A B))`

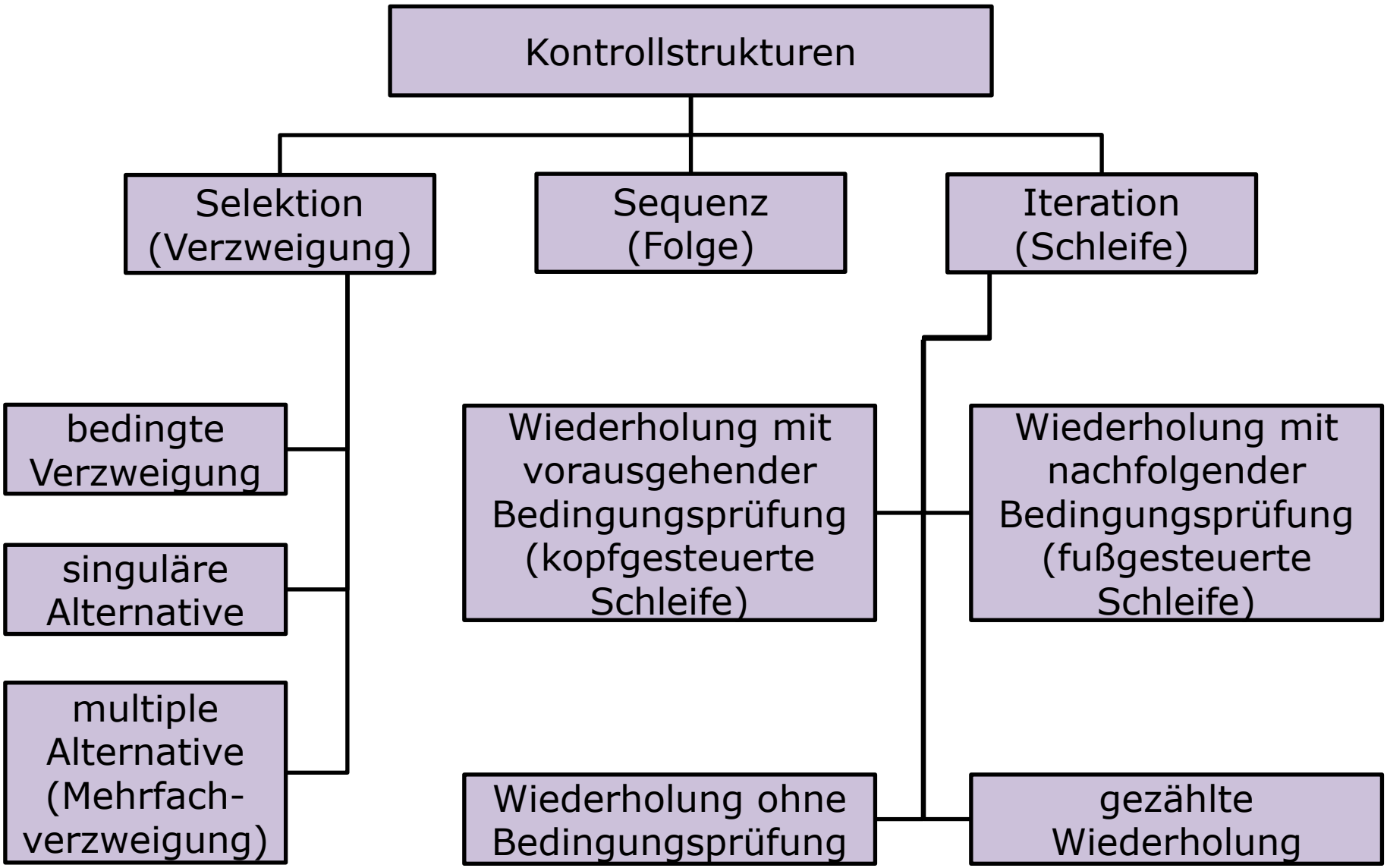
Programmiersprachen – Klassifizierung (4/4)

- objektorientierte Programmiersprachen
  - Weiterentwicklung der modularen Programmierung
  - Kapselung von Daten und dazugehörigen Methoden zu Objekten
  - Möglichkeiten der Formulierung allgemeiner Eigenschaften und Methoden (interne Funktionen) in Form der Klassen
  - Wiederverwendbarkeit durch Vererbung und Ableitung
  - ereignisgesteuerter Programmablauf
- mehr dazu im Modul 3IM-OOE-20

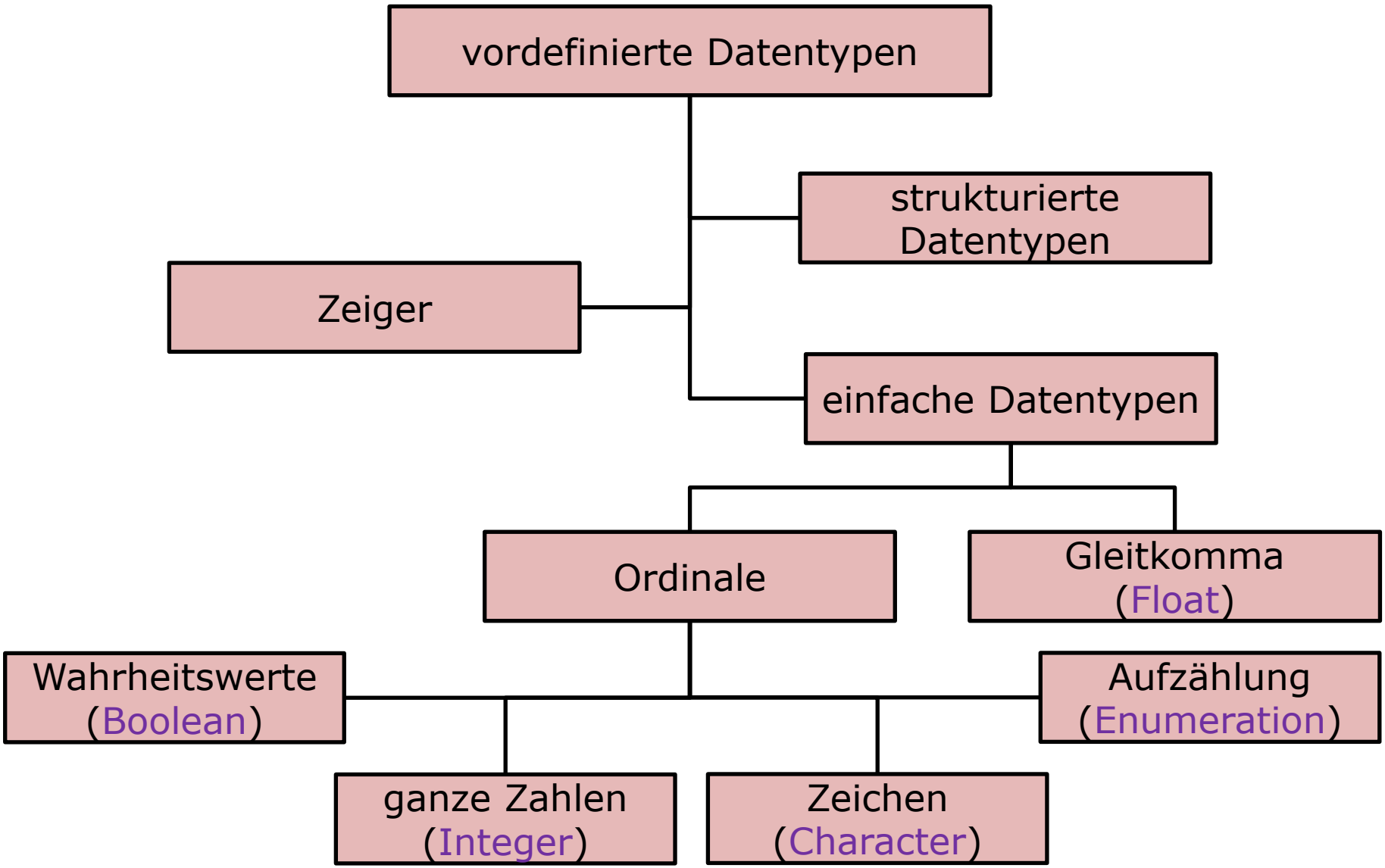
Programmiersprachen – Konzepte (1/3)



Programmiersprachen – Konzepte (2/3)



Programmiersprachen – Konzepte (3/3)



Programmiersprachen – Beschreibung (1/4)

Backus-Naur-Form (Backus-Normalform; BNF)

- künstliche Sprache zur Beschreibung von Daten und Abläufen
- Sprachelemente
  - Leeres Wort (ε)
  - Terminalsymbole (z.B. anzuzeigende Symbole und ε)
  - Konkatenation
  - Bezeichner in spitzen Klammern (<Bezeichner>)
  - Alternative ($Element_L$ | $Element_R$)
  - Zuweisung (::=)
  - Startelement (→) markiert Einstiegspunkt
  - Reihenfolgevorgabe wie in Mathematik (mit runden Klammern)
- vereinfachtes Beispiel:

```
→ <Palindrom>
  <Palindrom> ::= A(<Palindrom>)A | B(<Palindrom>)B | …
                | Z(<Palindrom>)Z | <Buchstabe> | ε
  <Buchstabe> ::= A | B | C | … | Z
```

Programmiersprachen – Beschreibung (2/4)

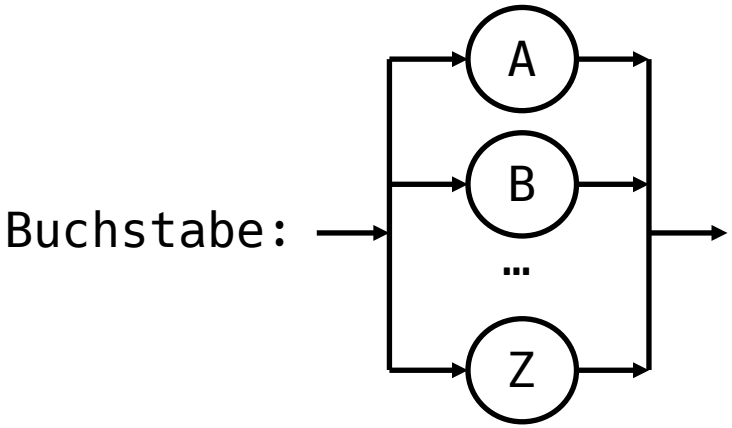
Erweiterte Backus-Normalform (EBNF)

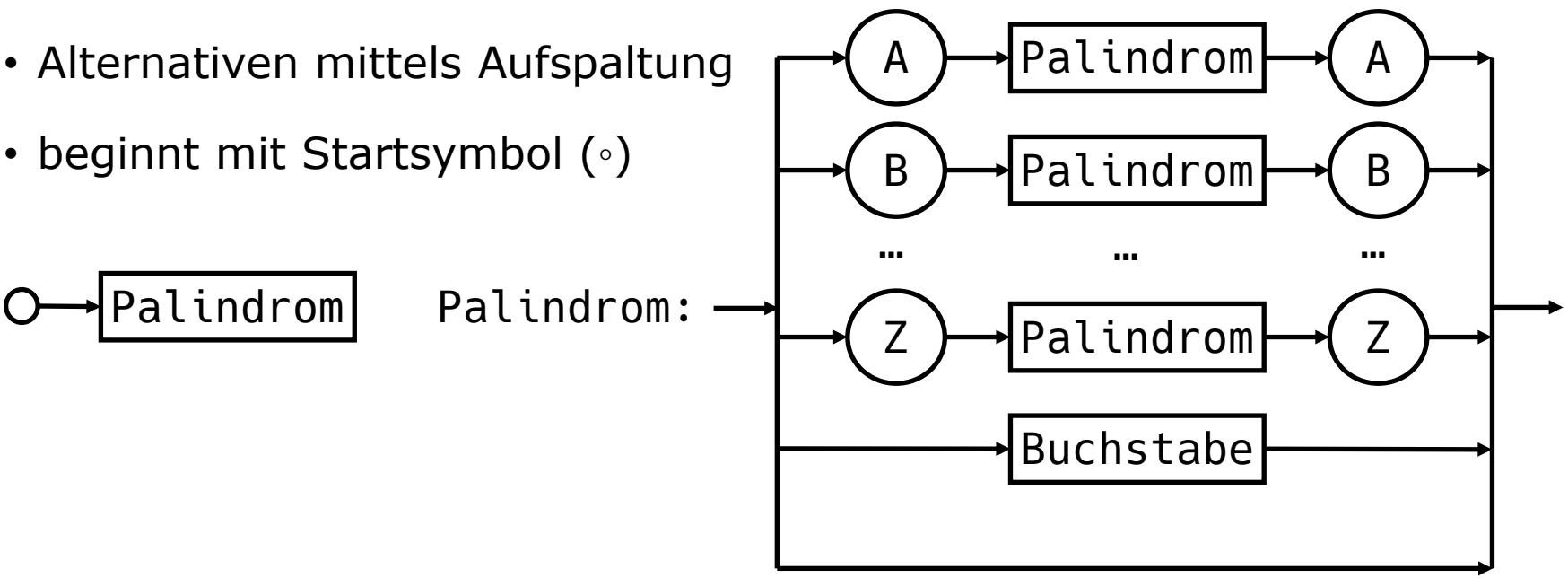
- dient der Darstellung kontextfreier Grammatiken (Typ-2-Sprachen)
- Sprachelemente der BNF
- zusätzlich:
  - optionales Element (d.h. einmal oder keinmal) ([…])
  - explizite Konkatenation (Konkatenationssymbol: ⊙; seltener: ,)
  - implizite Konkatenation bleibt erhalten (Konkatenationssymbol: Leerzeichen)
  - Wiederholung (auch keinmal) ({ … }; seltener: $\{ \ldots \}^*$)
  - Pflichtwiederholung (mindestens einmal) ($\{ \ldots \}^+$; seltener: $\{ \ldots \}_1$)
  - Variablen (nicht von <> oder " " eingeschlossene Zeichenfolgen)

Programmiersprachen – Beschreibung (3/4)

Syntaxdiagramm

- visuelle Darstellung von Elementen
- ähnlich EBNF
  - Bezeichner in Rechtecken
  - Terminalsymbole in Kreisen
  - Alternativen mittels Aufspaltung
  - beginnt mit Startsymbol (◦)

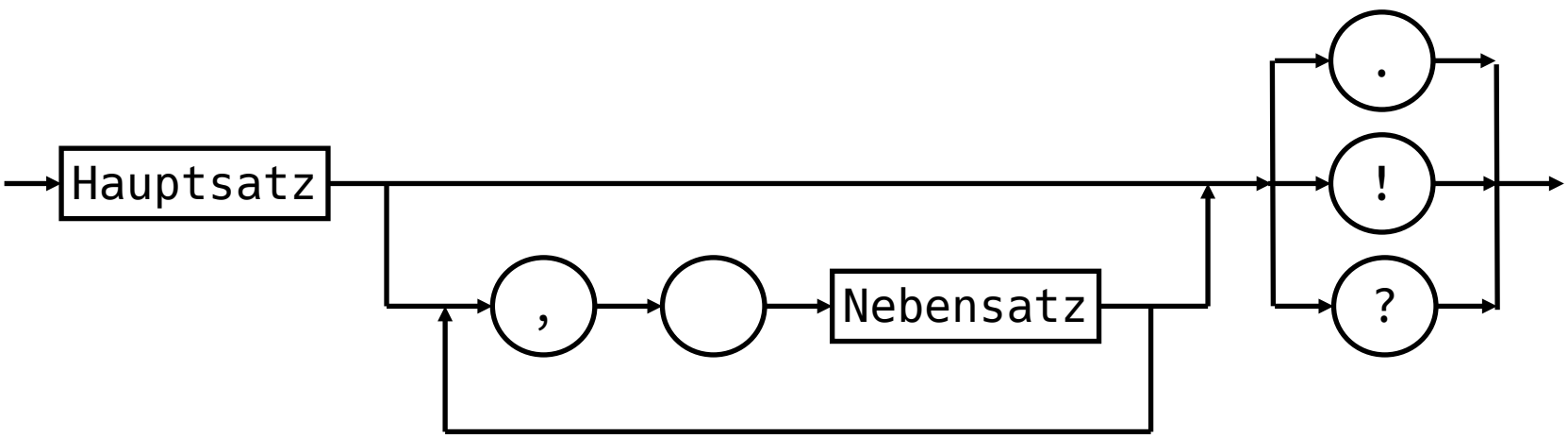




Programmiersprachen – Beschreibung (4/4)

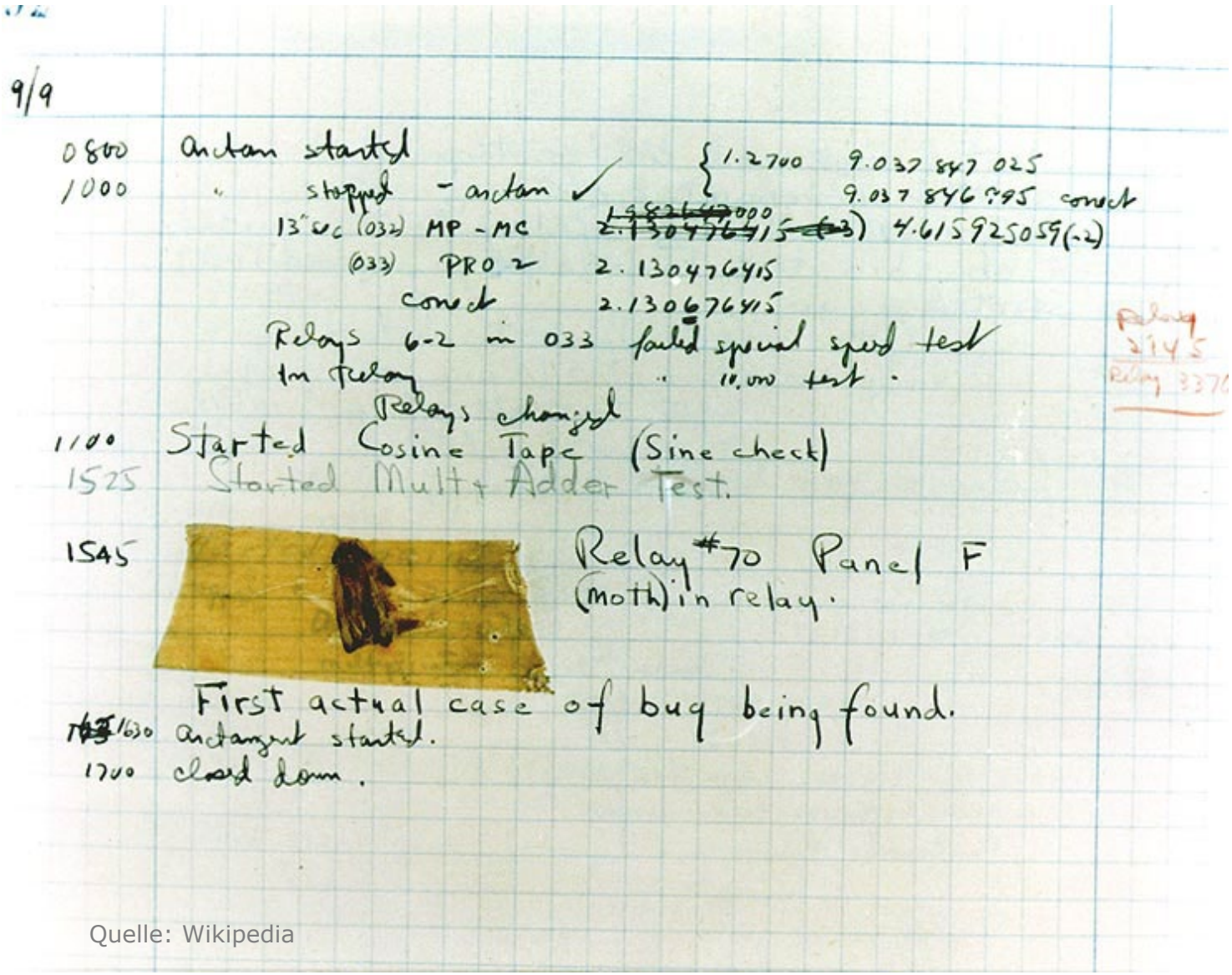
- EBNF und Syntaxdiagramme sind in der Ausdrucksstärke identisch
- Übersetzungsregeln erlauben gegenseitige Überführung
- Beispiel:

<Hauptsatz> ⊙ [{"," ⊙ " " ⊙ <Nebensatz>}$^{+}$] ⊙ ("."|"!"|"?")



ERSTER MODERNER „BUG“

1945:
Fehler im Relais #70 von Mark II, verursacht durch eine Motte (bug)



„Bug“-Begriff seit 1870er-Jahre nachweisbar

Quelle: Wikipedia

LITERATURANGABEN

D. Herrmann: *„Algorithmen Arbeitsbuch“,* ADDISON-WESLEY Verlag, ISBN 3-89319-481-9

S. Stein: *„Software Engineering Methoden“,* IBM Bildungszentrum NordWest, DOC SWE-ME SCRIPT 1992

H.-U. Karl, B. Speck: *„Grundlagen der Informatik“,* VMS Verlag Modernes Studieren Hamburg – Dresden GmbH, Bestellnummer: 1039 02 0

H. Schröder: *„Grundlagen der Programmierung“,* Herdt-Verlag für Bildungsmedien GmbH, PG99 30-0-07-02-01

J. Ziegenbalg: *„Algorithmen von Hammurapi bis Gödel“,* Spektrum Akademischer Verlag GmbH, ISBN 3-8274-0114-3

A. Solymosi / U. Grude: *„Grundkurs Algorithmen und Datenstrukturen“,* Vieweg Verlagsgesellschaft mbH, ISBN 3-528-05743-2