
I.2. Grundlagen von Programmiersprachen

- 1. Der Begriff „Informatik“
- 2. Syntax und Semantik von Programmiersprachen

1. Der Begriff „Informatik“

- **"Informatik"** = *Kunstwort* aus *Information* und *Mathematik*
 - Wissenschaft der Informationsverarbeitung mit großer Nähe zur Mathematik

- **Hauptaufgabe der Informatik**
 - *Entwicklung formaler, maschinell ausführbarer Verfahren* zur Lösung von Problemen der Informationsverarbeitung

- **Forderung der Durchführbarkeit mittels einer Maschine:**
 - Informationen müssen als *maschinell verarbeitbare Daten* dargestellt werden
 - Lösungsverfahren müssen bis *ins Detail* formal beschrieben werden.

Algorithmus

- **Ein Algorithmus ist ein Verfahren, welches**
 - in einem *endlichen* Text niedergelegt werden muss
 - *effektiv* (durch eine *Maschine*) *ausgeführt* werden kann
 - aus (direkt maschinell ausführbaren) Elementaroperationen besteht und *eindeutig* festlegt, welche Elementaroperation als nächstes auszuführen ist (*Determinismus*)
 - *Ein- und Ausgabe* ermöglicht, wobei jeder Eingabe genau eine Ausgabe zugeordnet wird (*Determiniertheit*)
- **Anzahl und Ausführungszeit der Elementaroperationen sind beschränkt.**
- **Ein Algorithmus (Programm) wird durch eine Maschine schrittweise ausgeführt**
 - Die ausführende Instanz muss die Vorschrift *interpretieren* und *korrekt* ausführen.
 - Ein Algorithmus *terminiert*, wenn er nach endlich vielen Schritten abbricht.

Deterministischer Algorithmus

Berechnung von $|x-y|$

1. Lies Eingaben x und y .
2. Falls $x \leq y$: Weiter mit Schritt 3.
Falls $x > y$: Weiter mit Schritt 4.
3. Berechne $a = y - x$.
Weiter mit Schritt 5.
4. Berechne $a = x - y$.
5. Gib a aus.

Indeterministischer Algorithmus

Berechnung von $|x-y|$

1. Lies Eingaben x und y .
Weiter mit Schritt 2 oder Schritt 3.
2. Berechne $a = x - y$.
Weiter mit Schritt 4.
3. Berechne $a = y - x$.
4. Falls $a \geq 0$: Gib a aus.
Falls $a < 0$: Gib $-a$ aus.

Fragen

- **Wie kann man aus einer Lösungsidee einen Algorithmus konstruieren?**
 - "schrittweise Programmentwicklung"

- **Wie zeigt man, dass ein Algorithmus tatsächlich das tut, was er tun soll?**
 - Verifikation: partielle Korrektheit
 - Terminierung

- **Wie "gut" ist ein Algorithmus?**
 - Speicherverbrauch, benötigte Zeit (*Effizienz*)
 - Aufwandsabschätzungen

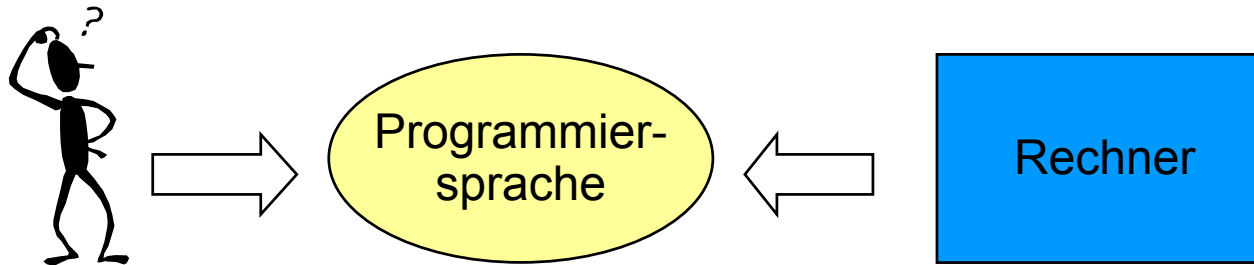
2. Syntax & Semantik von Programmiersprachen

Ein erstes Java-Programm:

```
public class Rechnung {  
  
    public static void main (String [] arguments) {  
  
        int x, y;  
        x = 10;  
        y = 23 * 33 + 3 * 7 * (5 + 6);  
        System.out.print ("Das Resultat ist ");  
        System.out.println (x + y);  
  
    }  
  
}
```

Programmiersprachen

- Die Programmiersprache bildet die **Schnittstelle** zwischen Mensch und Rechner



Beide haben unterschiedliche Anforderungen

- **Mensch**

- ◆ Erlernbarkeit
- ◆ Lesbarkeit
- ◆ Ausdrucksstärke

Höhere Programmiersprachen

A yellow thought bubble with a tail pointing towards the human requirements list, containing the text 'Höhere Programmiersprachen'.

- **Rechner**

- ◆ einfaches Übersetzen in Maschinsprache
- ◆ Generierung von effizientem Code

Maschinsprachen

A yellow thought bubble with a tail pointing towards the computer requirements list, containing the text 'Maschinsprachen'.

Kenntnis verschiedener Sprachen

- **Eigene Ideen bei der Software-Entwicklung können besser ausgedrückt werden**
- **Nötig, um in konkreten Projekten geeignete Sprache auszuwählen**
- **Erleichtert das Erlernen weiterer Programmiersprachen**
- **Nötig für den Entwurf neuer Programmiersprachen**

Übersicht

Imperative Sprachen

- Folge von nacheinander ausgeführten Anweisungen

■ Prozedurale Sprachen

- Variablen, Zuweisungen, Kontrollstrukturen

■ Objektorientierte Sprachen

- Objekte und Klassen
- ADT und Vererbung

Deklarative Sprachen

- Spezifikation dessen, was berechnet werden soll
- Festlegung, wie Berechnung verläuft durch Compiler

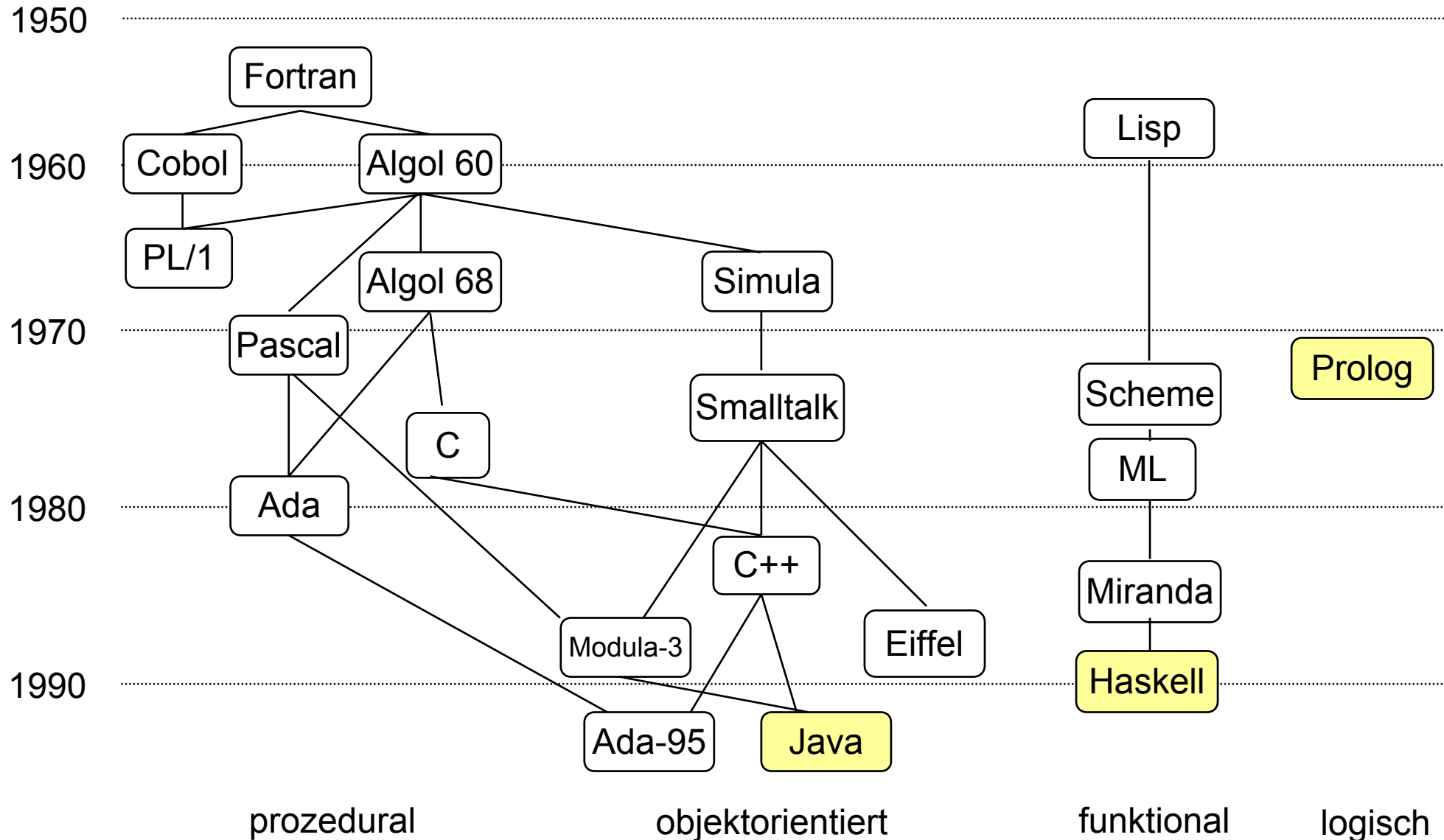
■ Funktionale Sprachen

- keine Seiteneffekte
- Rekursion

■ Logische Sprachen

- Regeln zur Definition von Relationen

Wichtige Programmiersprachen



Programmiersprachen - Definition

- Programmiersprachen sind Sprachen, deren Syntax und Semantik genau festgelegt ist.

- **Syntax:**
 - Definition aller *zulässigen Wörter / Programme*, die in einer Sprache formuliert werden können

- **Semantik:**
 - *Bedeutung* der zulässigen Wörter / Programme
 - Syntaktisch *falsche* Wörter / Programme haben *keine* Semantik

Alphabet, formale Sprache

■ Alphabet

- **nichtleere endliche** Menge von Zeichen („Buchstaben“, Symbolen)

■ Wort über einem Alphabet

- **endliche Folge** von Buchstaben, die auch **leer** sein kann (ϵ leere Wort)
- A^* bezeichnet die **Menge aller Wörter** über dem Alphabet A (inkl. dem leeren Wort)

■ Formale Sprache

- Sei A ein Alphabet. Eine (formale) Sprache (über A) ist **eine beliebige Teilmenge von A^*** .

■ Endliche Beschreibungsvorschrift für unendliche Sprachen

- **Grammatik**, die Sprache erzeugt, oder **Automat**, der Sprache erkennt

Beispiele

■ Beispiel 1

- $A_1 = \{0, 1\}$, $A_1^* = \{\varepsilon, 0, 1, 01, 10, 11, 000, 100, \dots\}$
- $L = \{0, 1, 10, 11, 100, 101, \dots\} \subseteq A_1^*$, die Menge der Binärdarstellungen natürlicher Zahlen (mit Null, ohne führende Nullen)

■ Beispiel 2

- $A_2 = \{(\, , \, +, \, -, \, *, \, /, \, a\}$, $A_2^* = \{\varepsilon, (\, , \, (+-a), \, (a^*a), \dots\}$
- die Sprache der korrekt geklammerten Ausdrücke $\text{EXPR} \subseteq A_2^*$:
 $\text{EXPR} = \{(((a))), (a + a), (a - a)^*a + a / (a + a) - a, \dots\}$

Grammatik - informell

- Definiert *Regeln*, die festlegen, welche Wörter über einem *Alphabet* zur Sprache gehören und welche nicht.
- Beispiel: Grammatik für "Hund-Katze-Sätze"

1	Satz	→	Subjekt Prädikat Objekt
2	Subjekt	→	Artikel Attribut Substantiv
3	Artikel	→	ϵ
4	Artikel	→	der
5	Artikel	→	die
6	Artikel	→	das
7	Attribut	→	ϵ
8	Attribut	→	Adjektiv
9	Attribut	→	Adjektiv Attribut
10	Adjektiv	→	kleine
11	Adjektiv	→	bissige
12	Adjektiv	→	große
13	Substantiv	→	Hund
14	Substantiv	→	Katze
15	Prädikat	→	jagt
16	Objekt	→	Artikel Attribut Substantiv

Grammatik - informell

■ Grammatik für "Hund-Katze-Sätze"

- durch diese Grammatik können z.B. die folgenden Sätze gebildet (abgeleitet) werden

"der kleine bissige Hund jagt die große Katze"

"die kleine Katze jagt der bissige Hund"

"das große Katze jagt der kleine große bissige kleine Katze"

- folgende "Sätze" werden nicht durch diese Grammatik gebildet

"die Katze der Hund"

"Katze und Hund"

"der Hund jagt die Katze die jagt Hund"

Grammatik - Definition - 1

■ Definition:

- Eine Grammatik G ist definiert durch ein *Viertupel* (N, T, P, S)

■ N : endliche Menge der *Nichtterminalsymbole (Variablen)*

- sind Symbole für syntaktische *Abstraktionen*
- **Beispiel:** Satz, Subjekt, Prädikat, Objekt, Artikel, Attribut, Substantiv, Adjektiv
- kommen nicht in den Wörtern der Sprache vor
- werden durch Anwendung der *Produktionsregeln* solange ersetzt, bis nur noch Terminalsymbole übrig sind

■ T : endl. Menge der *Terminalsymbole, disjunkt* mit N : $N \cap T = \emptyset$

- sind Zeichen des Alphabets, aus denen die Wörter der Sprache bestehen
- **Beispiel:** der, die, das, kleine, bissige, große, Hund, Katze, jagt

Grammatik - Definition - 2

■ P: endliche Menge von *Produktionsregeln* $x \rightarrow y$

- Regel $x \rightarrow y$ bedeutet, dass das Teilwort x durch das Teilwort y ersetzt werden kann
- $x \in V^* N V^*$, $y \in V^*$, wobei $V = N \cup T$ (Vokabular). D.h.: sowohl x als auch y können beliebige Nichtterminal- und Terminalsymbole enthalten, x enthält mindestens ein Nichtterminal.
- **Beispiel:** Prädikat \rightarrow jagt
der kleine bissige Hund Prädikat Objekt \Rightarrow
der kleine bissige Hund *jagt* Objekt

■ S: das *Startsymbol*

- ist ein spezielles Nichtterminalsymbol aus N , aus dem *alle Wörter* der Sprache mit Hilfe der Grammatik erzeugt werden
- **Beispiel:** Satz

Grammatik - Definition - 3

■ Ableitung

- Ableitungsprozess ist eine Relation " \Rightarrow " auf V^*
- Für $u, v, y \in V^*$ und $x \in V^*NV^*$ gilt
 - ◆ $uxv \Rightarrow uyv$ genau dann, wenn $(x \rightarrow y) \in P$

■ Die von der Grammatik **G** *erzeugte Sprache* ist:

- $L(G) = \{ w \mid w \in T^*, S \Rightarrow \dots \Rightarrow w \}$



w ist ableitbar aus dem Startsymbol

- D.h.: Jedes durch Anwendung der Regeln aus S erzeugbare Wort, *das nur aus Terminalsymbolen besteht*, gehört zu der Sprache $L(G)$
- zwei Grammatiken heißen *äquivalent*, wenn sie dieselbe Sprache erzeugen

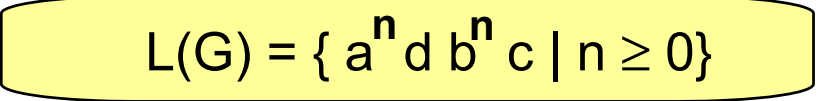
Kontextfreie Grammatik / Sprache

- **Produktionsregeln:** $A \rightarrow y$ $A \in N, y \in V^*$
d.h.: Links steht genau ein Nichtterminalsymbol
- **Wichtigste Klasse zur formalen Beschreibung der Syntax von Programmiersprachen.**
- **Es ist möglich, Automaten zu bauen, die Wörter einer kontextfreien Sprache erkennen (*Wortproblem*) und ihre syntaktische Struktur analysieren (Compilerbau)**
- **Notationen zur Darstellung kontextfreier Grammatiken**
 - *Syntaxdiagramme*
 - *Extended Backus-Naur-Form (EBNF)*

Grammatik - Beispiel

■ Sei $G = (N, T, P, S)$ mit

- $N = \{A, B\}$
- $T = \{a, b, c, d\}$
- $P = \{ A \rightarrow aBbc, B \rightarrow aBb, aBb \rightarrow d \}$
- $S = A$


$$L(G) = \{ a^n d b^n c \mid n \geq 0 \}$$

- G ist keine kontextfreie Grammatik, da die dritte Produktionsregel auf der linken Seite mehr als nur das Nichtterminalsymbol B enthält.

■ Ersetzt man in G die Produktionen P durch P' , dann ist G' kontextfrei. Es gilt $L(G) = L(G')$

- $P' = \{ A \rightarrow Bc, B \rightarrow aBb, B \rightarrow d \}$

EBNF

■ EBNF (Extended Backus-Naur-Form)

- kompaktere Repräsentation kontextfreier Grammatiken
- BNF erstmals benutzt zur Definition der Sprache *Algol-60*
- **EBNF-Notation**
 - ◆ = „definiert als“
 - ◆ (...|...) genau eine Alternative aus der Klammer muss kommen
 - ◆ [...] Inhalt der Klammer kann kommen oder nicht
 - ◆ { ... } Inhalt der Klammer kann n-fach kommen, $n \geq 0$
 - ◆ Terminalsymbole werden in " " eingeschlossen

Beispiel - Grammatik

1	Satz	→	Subjekt Prädikat Objekt
2	Subjekt	→	Artikel Attribut Substantiv
3	Artikel	→	ϵ
4	Artikel	→	der
5	Artikel	→	die
6	Artikel	→	das
7	Attribut	→	ϵ
8	Attribut	→	Adjektiv
9	Attribut	→	Adjektiv Attribut
10	Adjektiv	→	kleine
11	Adjektiv	→	bissige
12	Adjektiv	→	große
13	Substantiv	→	Hund
14	Substantiv	→	Katze
15	Prädikat	→	jagt
16	Objekt	→	Artikel Attribut Substantiv



Beispiel in EBNF

- Unsere einfache Grammatik für "Hund-Katze-Sätze" sieht in EBNF folgendermaßen aus:

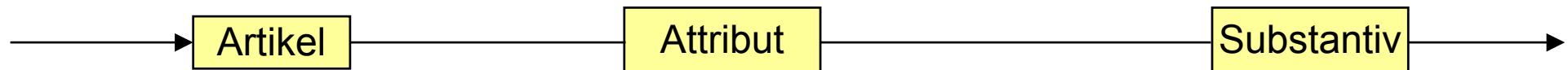
Satz	=	Subjekt Prädikat Objekt
Subjekt	=	Artikel Attribut Substantiv
Artikel	=	[("der" "die" "das")]
Attribut	=	{ Adjektiv }
Adjektiv	=	("kleine" "bissige" "große")
Substantiv	=	("Hund" "Katze")
Prädikat	=	"jagt"
Objekt	=	Artikel Attribut Substantiv

Syntaxdiagramme - 1

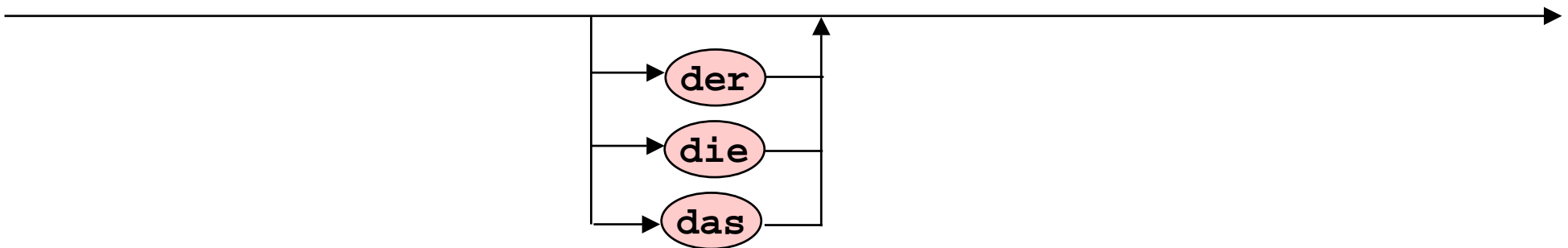
■ Syntaxdiagramme (beschreiben Produktionen *grafisch*)

- Nichtterminalsymbole sind Rechtecke 
- Terminalsymbole sind rund / oval 

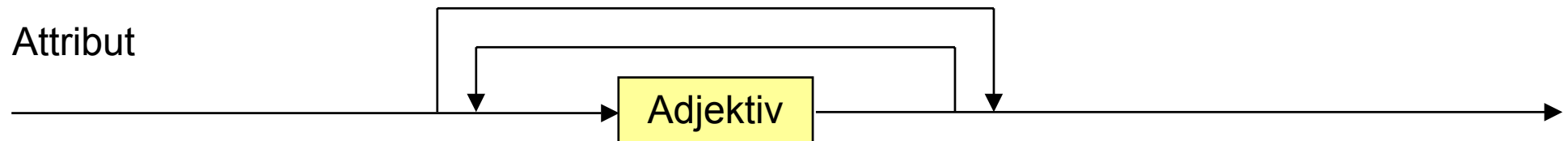
Subjekt



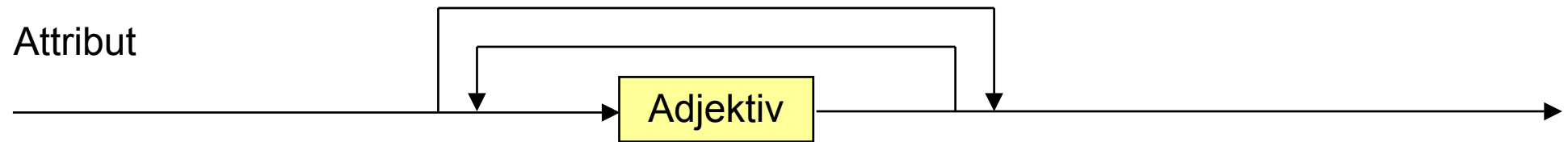
Artikel



Attribut



Syntaxdiagramme - 2



Alternative: **Rekursives** Syntaxdiagramm

