Technische Universität
Kaiserslautern

o1o1seda o1o1oo

software engineering dependability

Software Entwicklung 2

Übersetzerbau 1

Inhalt

- Aufgaben von Übersetzern
- Weitere Anwendungen von Übersetzern
- Arten von Übersetzern
- Grundlagen: Sprachdefinition
- Grammatik: BNF, EBNF, Syntaxdiagramme
- Produktionen, Ketten, Kettenmengen
- Satzform, Satz, Ableitung
- Chomsky-Hierarchie
- Aufbau von Compilern
- Aufgaben der Compiler-Komponenten: Analyseteil
- Aufgaben der Compiler-Komponenten: Syntheseteil
- Einpaß- vs. Mehrpaß-Compiler

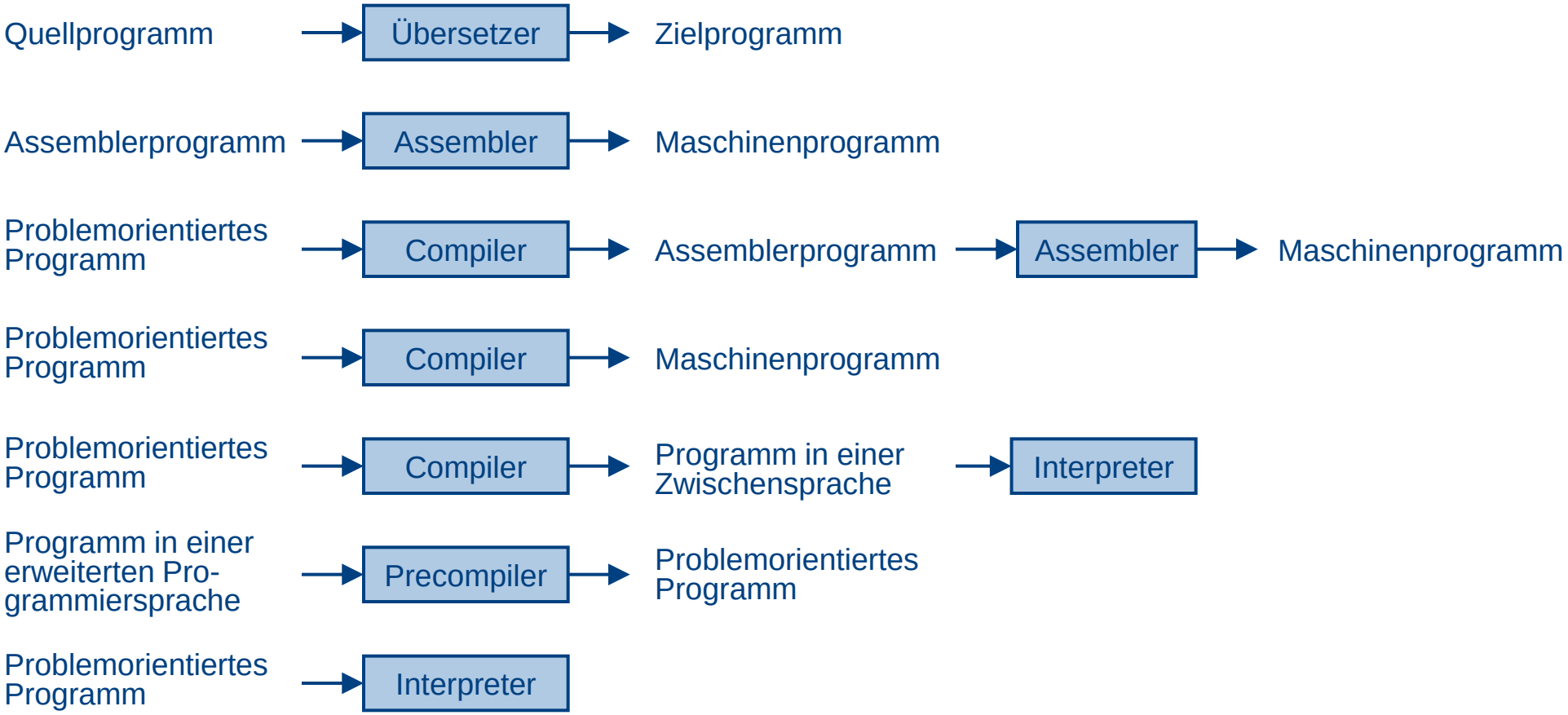
Lernziele

- Wissen, welche Arten von Übersetzern es gibt
- BNF, EBNF und Syntaxdiagramme anwenden können und insbesondere ineinander überführen können
- Die Begriffe Terminalsymbol, Nichtterminalsymbol, Kette, Ableitung und Sprache definieren können
- Die Chomsky-Hierarchie anwenden können
- Aufgaben der unterschiedlichen Teile eines Übersetzers kennen

- Die Aufgabe der Übersetzung von Texten einer Sprache A in eine andere Sprache B ist für natürlich gesprochene und geschriebene Sprachen allgemein bekannt.
- Analog ist es in der Informatik die zentrale Aufgabe eines Übersetzers, alle Sätze einer Quellsprache A, die Programme, in gleichbedeutende Sätze einer Zielsprache, die Zielprogramme, zu transformieren.
- Der Begriff Übersetzer ist dabei die übergeordnete Bezeichnung von Werkzeugen für diese Aufgabe. In Abhängigkeit von den Eigenschaften der Quellsprache und den Eigenschaften der Zielsprache werden spezialisierte Begriffe verwendet.
- Mit den gleichen Techniken können Softwarekomponenten realisiert werden, die z.B. Kommandosyntax an Kommandoschnittstellen analysieren.

- Formatierer wie *nroff* und *troff* in UNIX sind in Wirklichkeit Übersetzer, die Text und Formatierkommandos übersetzen
- Precompiler wie *equ (Formatierung von* Ausdrücken), *tbl* (Erzeugung von Tabellen), *pic* (Zeichnen von Bildern) erlauben eine Erweiterung von *nroff* und *troff*.
- Ein *silicon compiler* übersetzt die Layout- und Aufbauspezifikation für VLSI-Schaltkreise in eine VLSI-Schaltkreismaske.
- Abfrageprogramme für Datenbanken. z.B. geschrieben in SQL werden durch Übersetzer in physikalische Zugriffsbefehle auf die Datenbankdateien transformiert.
- Programme geschrieben in einer Roboterprogrammiersprache werden durch einen Compiler in eine Zwischensprache, z.B. IRDATA (Industrial Robot Data) übersetzt.

Compiler
Arten von Übersetzern



Beispiel:

- Regeln zur Bildung gültiger Sätze:
  - SATZ ::= SUBJEKT PRÄDIKAT.
  - SUBJEKT ::= "Peter" | "Siegfried".
  - PRÄDIKAT ::= "arbeitet" | "schläft".

- Gültige Sätze dieser Sprache:
  - Peter arbeitet
  - Siegfried arbeitet
  - Peter schläft
  - Siegfried schläft

- Ungültiger Satz dieser Sprache:
  - Peter arbeitet nicht

Beispiel (erweitert):

- Regeln zur Bildung gültiger Sätze:
  - SATZ ::= SUBJEKT PRÄDIKAT ZUSATZ.
  - SUBJEKT ::= "Peter" | "Siegfried".
  - PRÄDIKAT ::= "arbeitet" | "schläft".
  - ZUSATZ ::= "nicht" | ε.

- Gültige Sätze dieser Sprache:
  - Peter arbeitet
  - Siegfried arbeitet
  - Peter schläft
  - Siegfried schläft
  - Peter arbeitet nicht
  - Siegfried arbeitet nicht
  - Peter schläft nicht
  - Siegfried schläft nicht

Grammatik

- Viertupel aus
  - N: Vokabular der nicht-terminalen Symbole:
    z.B. SATZ, SUBJEKT, PRÄDIKAT
  - S: Startsymbol aus N: z.B. SATZ
  - T: Vokabular der terminalen Symbole:
    z.B. Peter, Siegfried, arbeitet, schläft
  - P: Menge der Produktionsregeln: siehe Beispiel

- N und T sind disjunkt!

- Vokabular V = N ∪ T

Grammatik
Die Backus-Naur-Form (BNF)

- 1960 von Backus und Naur zur Beschreibung von Algol 60 eingeführt:
- syntax ::= production syntax | ε.
- production ::= identifier "::=" expression ".".
- expression ::= term I expression "|" term.
- term ::= factor I term factor.
- factor ::= identifier | string.

- identifier ::= letter I identifier letter I identifier digit.
- string ::= stringhead """".
- stringhead ::= """" I stringhead letter.
- letter ::= "A" I ...I "Z".
- digit ::= "0" I... I "9".

Grammatik
Die erweiterte Backus-Naur-Form (EBNF)

- 1977 von Wirth zur Verbesserung der Lesbarkeit eingeführt:
- syntax ::= {production}.
- production ::= identifier "::=" expression ".".
- expression ::= term {"|" term}.
- term ::= factor {factor}.
- factor ::= identifier | string | "(" expression ")" | "[" expression "]" | "{" expression "}".

- identifier ::= letter {letter | digit}.
- string ::= """ {character} """.
- letter ::= "A" I ...I "Z".
- digit ::= "0" I... I "9".

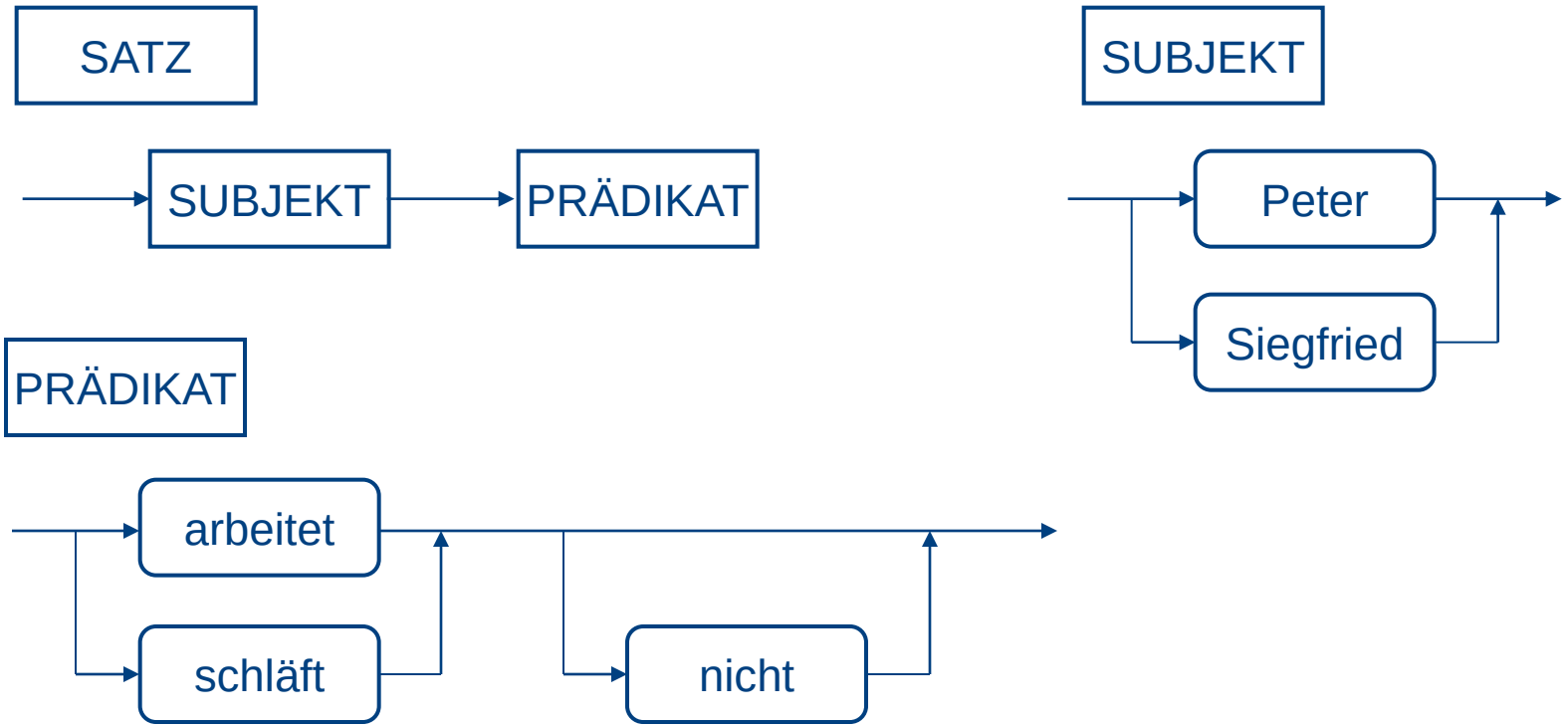
Grammatik
Die erweiterte Backus-Naur-Form (EBNF)

- ::= : „Zuweisungsoperator“ zwischen linker und rechter Seite einer Produktion
- | : Alternative
- { } : Wiederholung (kein mal, einmal und beliebig oft)
- [ ] : Option (kein mal oder einmal)
- ( ) : Klammerung

- Beispiel: Sprache, die die Sätze ε, a, aa, aaa, ... enthält.
  - BNF: S ::= A.
    A ::= aA | ε.
  - EBNF: S ::= A.
    A ::= {a}.

- Das erweiterte Beispiel in EBNF:
  - SATZ ::= SUBJEKT PRÄDIKAT.
  - SUBJEKT ::= "Peter" | "Siegfried".
  - PRÄDIKAT ::= ("arbeitet" | "schläft") ["nicht"].

- Anstelle von Produktionen in BNF oder EBNF können auch Syntaxdiagramme angegeben werden:



Grammatik
Produktionsregeln

- Jedes Syntaxdiagramm beschreibt eine Produktionsregel der Sprache
  - Die Produktionsregeln beschreiben, welche Symbolfolgen der Sprache legal bzw. erlaubt sind
- Notationen
  - Syntaxdiagramme
  - Erweiterte Backus Naur Form (EBNF)
  - Backus Naur Form (BNF).

- (Symbol-)Kette: Endliche Folge σ von Symbolen über einem Vokabular V
  - σ_1 = BeginSym ReadSym LParen <idlist> RParen SemiColonEndSym EofSym
  - σ_2 = BeginSym <StatementList> EndSym EofSym
- Eine Kette kann aus nicht-terminalen und terminalen Symbolen bestehen.

- Abkürzungen
  - a^n: Kette aus n gleichen Symbolen a
    - a^3 = aaa
    - ε leere Kette, d.h. die Kette aus null Symbolen
  - a^+: Menge $\{a^n : n \geq 1\}$
    - $a^+ = \{a, aa, aaa, aaaa, ..\}$
  - a^* : Menge $\{a^n : n \geq 0\}$
    - $a^* = \{\varepsilon, a, aa, aaa, aaaa, ..\}$.
  - Offensichtlich ist $a^* = a^+ \cup \{\varepsilon\}$.

Grammatik
Kettenmengen

- V^+: Menge aller nichtleeren Ketten, die sich aus den Symbolen von V bilden lassen
  - $V^+ = V \cup V_2 \cup ...$
- V^* : Menge aller Ketten, einschließlich der leeren Kette, die sich aus den Symbolen von V bilden lassen
  - $V^* = \{\varepsilon\} \cup V^+$

- Die Menge V ist immer endlich
- Die Mengen a^+, a^*, V^+, und V* sind immer unendlich

- Für V= {a, b, c} sind
  - V^+ = {a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa,...}
  - V^* ={ ε ,a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa,...}

- Sprache = Menge von Symbolketten, die von der Syntax als wohlgeformt definiert sind.
- Produktionsregeln beschreiben Ersetzungen von Symbolketten durch Symbolketten
  - Ausgangspunkt: Startsymbol S
  - Wiederholtes Ersetzen gemäß den Regeln der Grammatik durch Symbolketten, bis eine Symbolkette entsteht, die nur noch terminale Symbole enthält.

- Beginn mit dem Syntaxdiagramm **Start** bzw. der Produktion, auf deren linker Seite das Nichtterminal-Symbol **Start** vorkommt
- Alle Syntaxdiagramme bzw. Produktionsregeln solange anwenden, bis nur noch terminale Symbole vorhanden sind
- Jede Sequenz terminaler Symbole, die so erzeugt werden kann, ist gültig
- Umgekehrt gilt, dass jede Sequenz terminaler Symbole, die nicht durch eine Sequenz von Ersetzungen nicht-terminaler Symbole erzeugt werden kann, illegal ist.

- Rekursiv anwendbare Analyseroutinen, die durch den Syntaxbaum absteigen, den sie bei der Programmbearbeitung erkennen
  - Einfachste Syntaxanalyse-Technik
  - wird in vielen Compilern verwendet

- Grundidee
  - Zu jedem nicht-terminalen Symbol existiert eine (gleichnamige) Analyseprozedur, die jede terminale Symbolfolge erkennt, die durch das nicht-terminale Symbol erzeugt werden kann.
  - In einer Analyse-Prozedur können sowohl nicht-terminale als auch terminale Symbole auftauchen.

Grammatik
Satzform, Satz, Ableitung

- Satzformen
  - Das Startsymbol S selbst und alle Symbolketten, die aus S durch die Anwendung der Produktionen hervorgehen
- Sätze
  - Satzformen, die nur aus terminalen Symbolen bestehen

- Sind α und β zwei Satzformen und geht β aus α durch Anwendung einer Produktion hervor, dann $\alpha => \beta$. (Ableitung)

- Direkte Ableitung
  - Eine Kette α produziert direkt eine Kette β
    - geschrieben $\alpha => \beta$,
  - wenn es irgendwelche Ketten ω_1 und ω_2 gibt, so daß
    - $\alpha = \omega_1 \sigma \omega_2$
    - $\beta = \omega_1 \tau \omega_2$ und
    - die Produktion $\sigma ::= \tau$ zur Grammatik gehört
  - β heißt dann eine direkte Ableitung von α

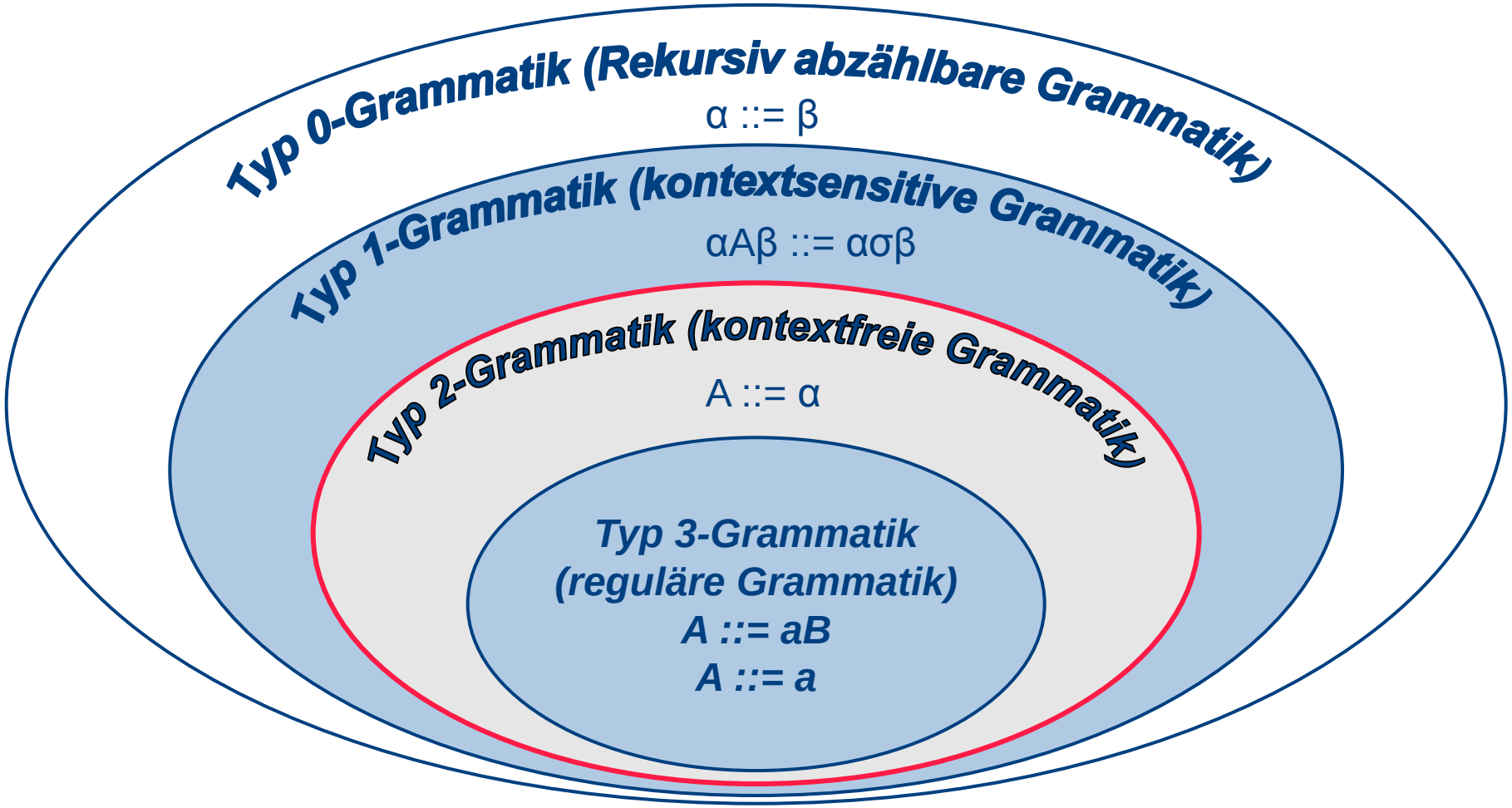
- Mehrfache Ableitung
  - Eine Kette α produziert eine Kette β
    - geschrieben $\alpha =>^+ \beta$,
    - wenn es eine Folge direkter Ableitungen $\alpha = \omega_0 => \omega_1 => \omega_2 => ... => \omega_n = \beta$ mit $n \geq 1$ gibt.
  - Diese Folge heißt mehrfache Ableitung der Länge n

- Wenn G eine Grammatik mit dem Startsymbol S ist, dann wird eine Kette α Satzform genannt, wenn $S =>^* \alpha$ gilt.
- Ein Satz ist eine Satzform, die nur aus terminalen Symbolen besteht.
- Eine Sprache L(G) ist die Menge aller Sätze, die sich aus dem Startsymbol ableiten lassen:

$$L(G) = \{ \alpha \mid S =>^+ \alpha \wedge \alpha \in T^* \}$$

- Die Chomsky-Hierarchie ist eine Aufteilung von Sprachen in Klassen:
- Reguläre Sprachen (Typ 3):
  - Können durch endliche Automaten dargestellt werden
  - Werden durch Scanner analysiert
  - Dürfen nur Produktionen der folgenden Form enthalten:
    $A \rightarrow a$ und $A \rightarrow a\ B$ bzw. $A \rightarrow a$ und $A \rightarrow B\ a$.
    (Kleinbuchstaben stellen Terminalsymbole dar, Großbuchstaben repräsentieren nichtterminale Symbole)
- Kontextfreie Sprachen (Typ 2)
  - Können durch Kellerautomaten dargestellt werden
  - Werden durch Parser analysiert
  - Auf der linken Seite jeder Produktion muß stets genau ein Nichtterminal-Symbol stehen,
    z.B.: $A \rightarrow a\ B\ b$
  - Die rechte Seite darf aus einer beliebigen Kombination von terminalen und nichtterminalen Symbolen bestehen

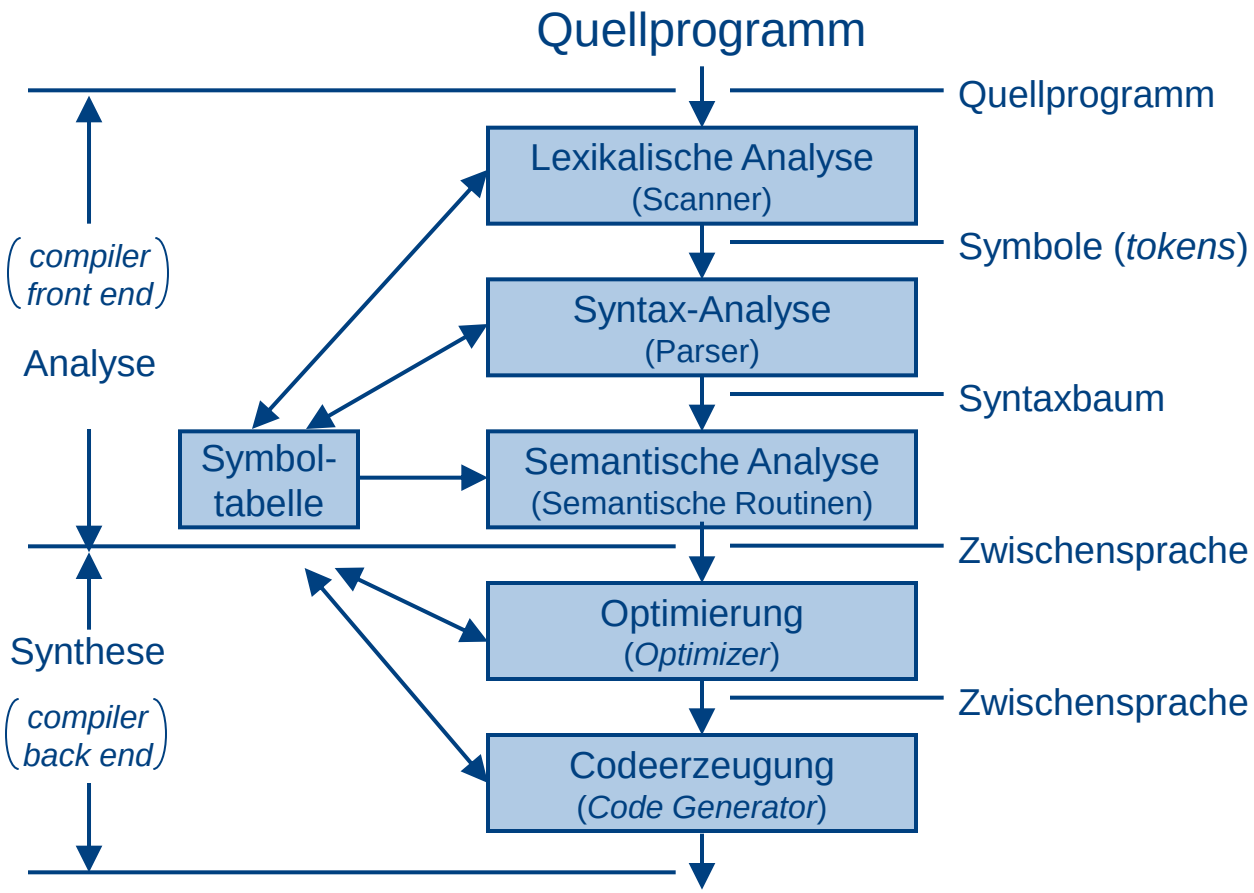
- Kontextsensitive Sprachen (Typ 1)
  - Produktionen haben die Form $\alpha\ \gamma\ \beta \rightarrow \alpha\ \sigma\ \beta$ mit $\alpha,\beta\ \in T^*$, $\sigma \in V^+$, $\gamma \in N^+$
  - Die rechte Seite von Produktionen muß mindestens so lang sein wie die linke Seite (längenmonoton)
  - Auf der rechten Produktionsseite darf nur dann das leere Symbol (e) stehen, wenn die linke Produktionsseite das Startsymbol S ist, und dieses auf keiner rechten Produktionsseite in der Grammatik auftritt: $S \rightarrow e$
- Rekursiv aufzählbare Sprachen (Typ 0)
  - Jede wohlgeformte Grammatik, in der die Terminalsymbole, Nichtterminal-Symbole und das Startsymbol definiert sind, ist vom Chomsky-Typ 0



α = *beliebige Kette aus N und T*

- Für den Compilerbau sind nur reguläre und kontextfreie Sprachen interessant
- Reguläre Grammatiken:
  - Definition der Bezeichner und Schlüsselworte, sowie von Konstanten und anderen Symbolen der Sprache
  - Scanner: Zerlegung des Eingabetextes in Symbole
- Kontextfreie Grammatiken:
  - Festlegung der Regeln, wie aus Symbolen korrekte Sätze der Sprache gebildet werden
  - Parser: Syntaxanalyse

- Kontextsensitivität: Übersetzung eines deutschen Textes ins Englische
- *Ich mag Montage nicht.*
- Version 1: *I don‘t like mondays.*
- Version 2: *I don‘t like assembly.*



Aufgaben der Compiler-Komponenten: Analyseteil

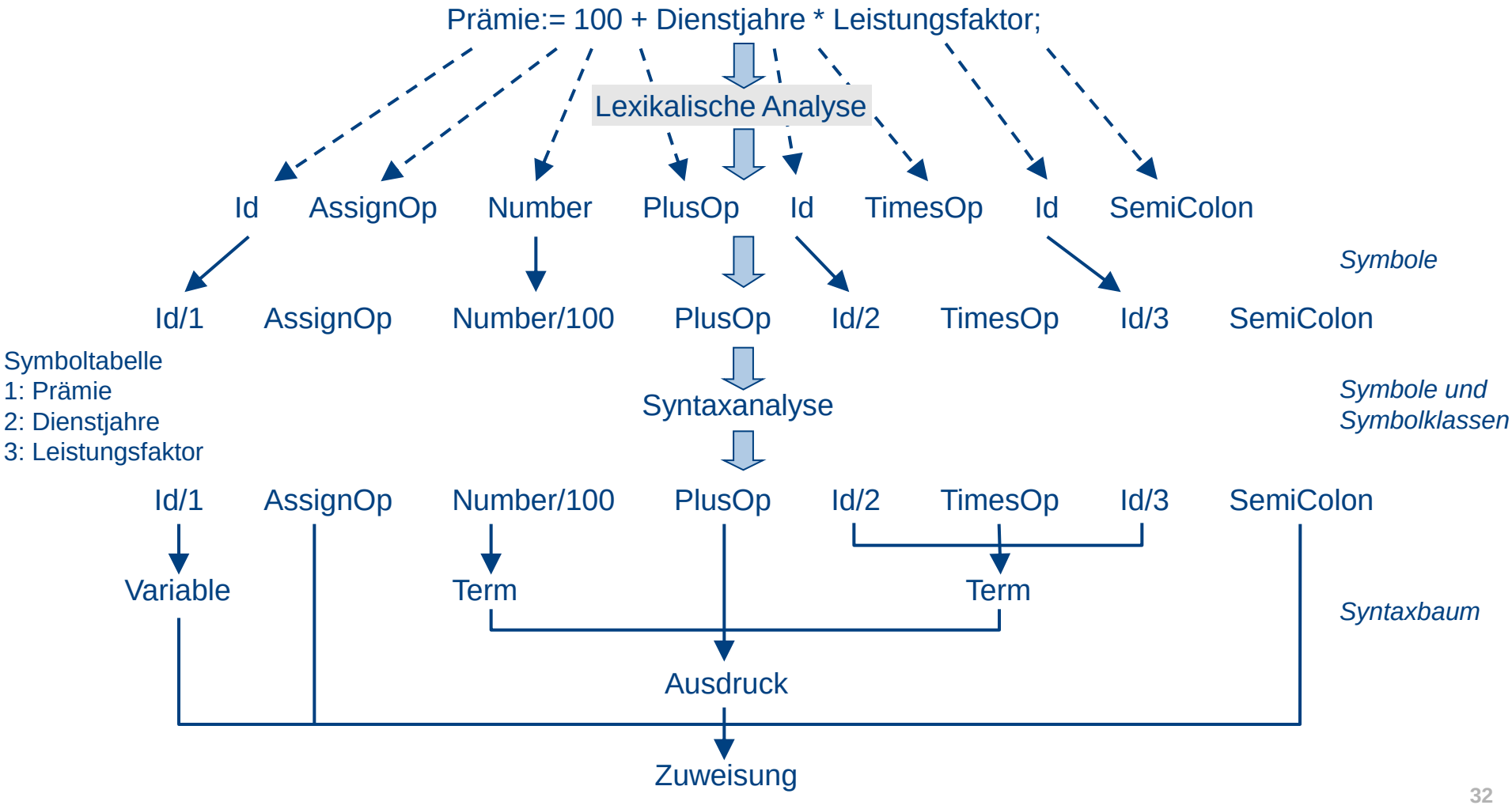
- **Scanner: Durchführung der lexikalischen Analyse:**
  - Das Quellprogramm wird zeichenweise gelesen. Die individuellen Zeichen werden zu Symbolen *(tokens)* zusammengefaßt, die die Eingabe für den Parser darstellen.
  - Beispiele für Symbole sind Bezeichner, Zahlen, Operatoren, Begrenzer wie Klammern und Semikolon.

- **Parser: Durchführung der syntaktischen Analyse:**
  - Das aus Symbolen bestehende Programm wird in seine grammatischen Bestandteile zerlegt und die Struktur als Syntaxbaum dargestellt.

Aufgaben der Compiler-Komponenten: Analyseteil

- Semantische Routinen: Durchführung der semantischen Analyse:
  - Überprüfung der statischen Semantik jedes Sprachkonstrukts, wie Gültigkeitsbereiche von Bezeichnern, Korrespondenz zwischen Deklaration und Anwendung von Bezeichnern, Typverträglichkeit von Operatoren in Ausdrücken. Ist das Konstrukt semantisch korrekt, so wird es in eine Zwischensprache transformiert.
  - Die semantische Analyse kann mit der Syntaxanalyse gemeinsam stattfinden. In diesem Fall verschmelzen diese beiden Analyseteile zu einem Analyseteil.
  - Erfolgt die semantische Analyse separat, dann wird der Syntaxbaum, das Ergebnis der Syntaxanalyse, bei der semantischen Analyse durch semantische Informationen ergänzt.
  - Als Ergebnis liefert der Analyseteil des Compilers,
    - die Feststellung, ob das Quellprogramm korrekt ist
    - eine speicherinterne Darstellung des Quellprogramms, die für den Syntheseteil gut geeignet ist,
    - speicherinterne Tabellen, die für die weitere Verarbeitung der Zwischendarstellung benötigt werden.

Beispiel: Lexikalische und syntaktische Analyse

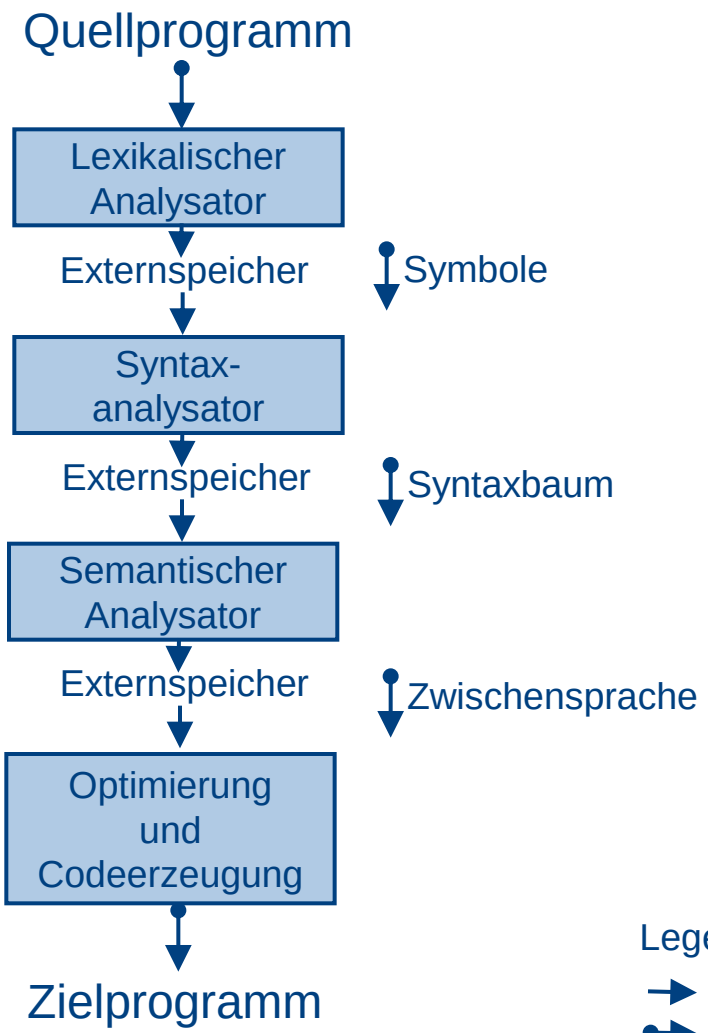


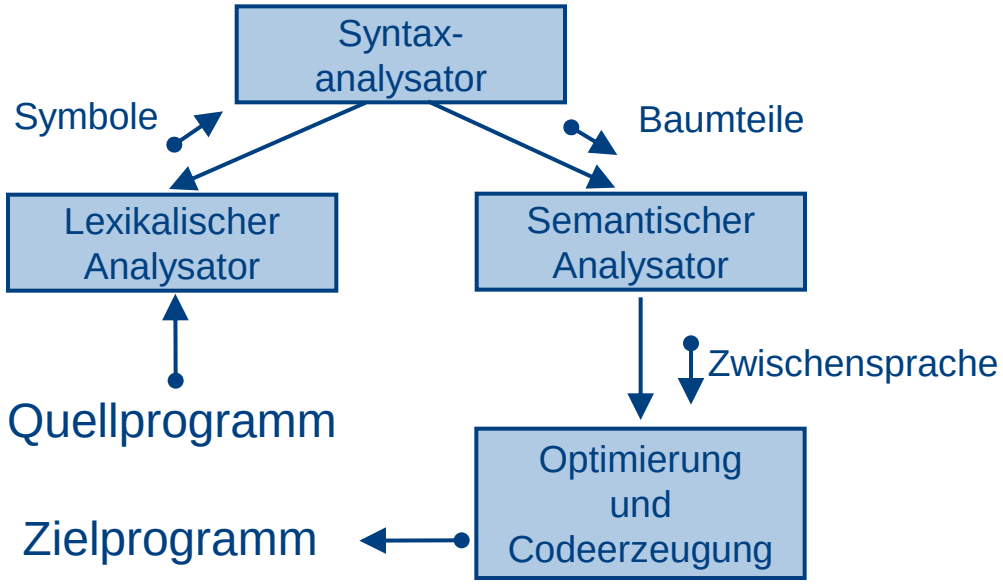
Aufgaben der Compiler-Komponenten: Syntheseteil

- Optimierer:
  - Transformation des Programms in der Zwischensprache in ein funktional äquivalentes Programm in der Zwischensprache, das hinsichtlich bestimmter Kriterien (z.B. Geschwindigkeit) verbessert ist.
  - Der Optimierer kann bei vielen Compilern auch deaktiviert werden.
- Code-Generator:
  - Erzeugung des Zielprogramms aus der Zwischendarstellung.
- Symboltabelle:
  - Sowohl der Analyseteil als auch der Syntheseteil des Compilers greifen auf die sogenannte Symboltabelle zu. In der Symboltabelle werden die Bezeichner mit ihren Attributen gespeichert.
  - Falls keine Optimierung gefordert wird, so kann die Code-Generierung in die semantischen Routinen integriert werden (keine Zwischendarstellung)

- Mehrpaß-Compiler:
  - Nacheinanderausführung der in der statischen Compilerstruktur aufgeführten Komponenten
  - Man spricht dann von einem Mehrpaß-Compiler, da eine Komponente nach der anderen ausgeführt wird. Als Ergebnis jeder Komponente entsteht ein Programm in einer Zwischensprache, das auf einen externen Speicher geschrieben und vom nächsten Paß wieder gelesen wird
- Einpaß-Compiler:
  - Zeitlich verzahnte Ausführung der einzelnen Compilerkomponenten
  - Der Parser ruft den Scanner auf, wenn er das nächste Quellsymbol benötigt. Er ruft die semantischen Routinen auf, wenn er eine syntak- tisch korrekte Konstruktion an sie weitergeben will.

Einpaß- vs. Mehrpaß-Compiler





Legende:
- Steuerfluss
- Datenfluss