

Grundlagen des Übersetzerbaus

WS 2013/14



Bedeutung einiger Folienmarkierungen

- Im Folientitel, z.B.

Organisatorisches (1/2)

Erste von zwei Folien zum Thema „Organisatorisches“

Organisatorisches (1a/2)

Zusätzliche Folie zu Folie „Organisatorisches (1/2)“

- Auf der Folie

Symbole

Begriffsdefinition/Begriffsklärung



*(meist ausgeblendete) Folie mit nützlichen Zusatz-Informationen;
Bezug zur „Quell-Folie“ über den jeweiligen Folientitel*

Methoden 

Begriff der im Folgenden (ggf. auf nachfolgender Folie) näher erläutert wird

klassifizieren

Hyperlink zu Folie, die weitere Informationen zum markierten Begriff enthält

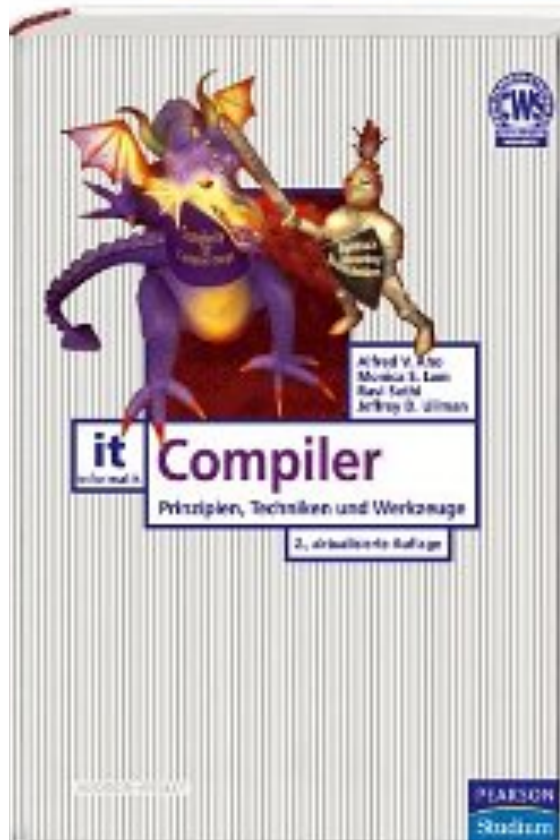


Hyperlink zu Folie, die weitere Informationen zum nebenstehenden Thema enthält



Hauptsächliche Hintergrund-Literatur

- R. Wilhelm, D. Maurer: Übersetzerbau. 2. Aufl. Springer 1997
- A. Aho, S. Lam, R. Sethi, J. Ullman: Compiler. 2. Aufl. Pearson Studium 2008
- R. Wilhelm, H. Seidl: Übersetzerbau – Virtuelle Maschinen. Springer 2007



Themen

- Grundprinzipien der Sprachübersetzung
- Schwerpunkt: Übersetzung **imperativer** Programmiersprachen

Inhalt (insgesamt)

- Einführung und Motivation
- Kern des Übersetzungsprozesses
- Lexikalische Analyse
- Syntaktische Analyse
- Semantische Analyse
- Codeerzeugung für reale Maschinen

Fortsetzung (im nächsten Semester): Übersetzung anderer Sprachkonzepte

- Funktionale Sprachen
- Logische Sprachen
- Objektorientierte Sprachen

Inhalt

- Motivation und Begriffsbildungen
- Konzeptionelle Übersetzerteilaufgaben
- Reale Übersetzerstrukturen

Lernziele

- Die Bedeutung des Übersetzerbaus erklären können
- Den zentralen Begriff „Übersetzer“ und damit verwandte Begriffe und ihre jeweiligen Zusammenhänge beschreiben können
- Die prinzipiellen Arbeitsschritte eines Übersetzers wiedergeben können, sowie beschreiben können, wie diese in realen Übersetzern umgesetzt werden

Prinzipielle Bedeutung des Übersetzerbaus

- Ohne höhere Programmiersprachen und Übersetzer keine „vernünftige“ Kommunikation zwischen Mensch und Rechner

Gilt sinngemäß auch für alternative Kommunikationsformen (z.B. Spracheingabe)

Beispiel: ggt-Algorithmus (in Turbo Pascal)

```
PROGRAM ggt;
var i, j, temp: integer;
BEGIN
  i := 15;    j := 21;
  WHILE (i <> 0) DO
  BEGIN
    IF (i < j) THEN
    BEGIN
      temp := i;
      i := j;
      j := temp;
    END;
    i := i - j;
  END;
  Write ('Größter gemeinsamer Teiler ist ', j);
END.
```

Beispiel (Forts.): ggt-Algorithmus (in ausführbarem Code, Ausschnitt)

```

00000000: 4D 5A 70 00 06 00 0E 00 06 00 25 04 25 F1 E3 00  MZp.....%.%ã.
00000010: 00 40 00 00 20 00 00 00 1C 00 00 00 23 00 00 00  .@.....#...
00000020: 2D 00 00 00 79 00 00 00 E7 00 00 00 EE 00 00 00  -...y...ç...i...
00000030: E6 00 00 00 D6 00 00 00 01 00 0A 00 1B 01 0A 00  æ...Ö.....
00000040: 10 03 0A 00 61 07 0A 00 7C 07 0A 00 C6 07 0A 00  ...a...|...Æ...
00000050: A6 07 0A 00 00 00 00 00 00 00 00 00 00 00 00 00  |.....
00000060: 1F 47 72 F6 DF 74 65 72 20 67 65 6D 65 69 6E 73  .Größter gemeins
00000070: 61 6D 65 72 20 54 65 69 6C 65 72 20 69 73 74 20  amer Teiler ist
00000080: DC 00 00 0A 00 55 EB D5 31 2B DC 2D 02 0A 00 C3  Ü...UeÖ1+Ü-...Ã
00000090: 06 52 00 A4 00 C3 06 54 00 A7 00 E2 3E 52 00 00  .R.¤.Ã.T.$.â>R..
000000A0: 74 27 ED 52 00 3B 06 54 00 7D 12 ED 52 00 FA 56  t'iR...T.}.iR.úV
000000B0: 00 ED 54 00 FA 52 00 ED 56 00 FA 54 00 ED 52 00  .iT.úR.iV.úT.iR.
000000C0: 2B 06 54 00 FA 52 00 D9 CA 2B 58 01 1E 57 2B 00  +.T.úR.ÜE+X..W+.
000000D0: 00 0E 57 31 2B 50 DC 70 06 0A 00 ED 54 00 D6 52  ..W1+PÜp...iT.ÖR
000000E0: 50 31 2B 50 DC B0 06 0A 00 DC A6 05 0A 00 DC E6  P1+PÜ^...Ü|...Üæ
000000F0: 02 0A 00 5D 31 2B DC 16 01 0A 00 00 00 00 00 00  ...]1+Ü.....
00000100: A6 F8 00 C4 2B EE 06 38 00 33 DD DE DE 08 DE E1  |ø.Ã+i.8.3Ýbb.bá
00000110: 00 EF 2D 05 13 00 A6 04 CB DE EE CA 03 2D FA 0A  .i-...|.ÉpiÉ.-ú.
00000120: 00 FA 0C 00 03 06 04 00 FA 0E 00 FA 18 00 FA 1C  .ú.....ú.ú.ú.
00000130: 00 FA 24 00 C4 06 38 00 26 ED 02 00 FA 20 00 C3  .ús.Ã.8.&i.ú.Ã
00000140: 06 2A 00 CD 00 EE 0E 2C 00 2B 58 02 A5 39 02 A6  *.Í.i...+X.¥9.|
00000150: 13 00 C9 B3 2E BC A6 35 2D 21 EB 1D EE 45 02 E2  ..É³.¼|5-!è.iE.â
00000160: C3 04 D4 B4 1E 0E 1F A6 0C 01 A9 00 25 2D 21 A6  Ã.Ö'...|...@.%-!!
00000170: 13 01 A9 23 25 2D 21 A6 A6 00 A9 24 25 2D 21 A6  ..@#%-!!!@%-!!
00000180: 04 01 A9 3F 25 2D 21 1F A9 58 00 1E 50 1E 50 A9  ..@?%-!@X..P.P@
00000190: 63 02 0E 50 0E DE 4E 02 0E DE 2B 02 A9 58 01 1E  c..P.pN..p+.@X..
000001A0: 50 1E 50 A9 63 02 0E 50 0E DE 3A 02 0E DE A6 02  P.P@c..P.p:..p|.
000001B0: 2D 33 2B A3 5B C7 FE A4 53 D8 A3 59 C7 D5 AD C7  -3+£[çbªS0£YÇÖ-Ç
000001C0: B2 AD 74 0E 40 C7 A4 AD 53 D8 A3 59 C7 D5 AD 74  ²-t.@Çª-S0£YÇÖ-t
000001D0: 01 40 F3 4C 00 2B 33 2B 2D 02 00 B9 E2 2D 06 58  .@óL.+3+-...¹â-.X
000001E0: E2 FE 1F FC C3 FB 00 C7 B3 39 73 03 2B A0 A0 57  áb.úÃú.Ç³9s.+ W
000001F0: A6 54 2D 21 EF FD C7 4E 16 01 58 5B 59 5A 5E 5F  |T-!iýÇN..X[VZ^
00000200: 5D 1F 07 A4 A9 F0 00 E2 2D 06 D9 03 A9 2B 00 59  ]..¤@ð.â-.Ü.®+.Y
00000210: 5B D9 07 A9 A0 00 33 2B 33 A6 A6 F8 00 C4 2B B9  [Ü.® .3+3||ø.Ã+¹
00000220: FA 32 00 EF 2D 0B 2B 74 3D ED 10 00 0B 2B 74 2F  ú2.i-.+t=i...+t/
00000230: C4 2B 26 ED 10 00 0B 2B 74 1B 2B 2B 77 17 B8 CF  Ã+&i...+t.++w.Í
00000240: 3D 00 10 73 10 A6 10 00 B8 D4 03 2D 72 07 26 3B  =...s.|...Ö.-r.&
00000250: 06 08 00 72 06 26 ED B6 00 D9 D0 EF 2B EE 2B 2B  ..r.&i¶.ÜDi+i++
00000260: 1E 38 00 E2 D9 10 EB 0E 34 00 EB 1E 36 00 2D 1E  .8.âÜ.è.4.è.6.-.
00000270: 2F 00 EF 2B 0B 2B 74 13 23 2B FA 2F 00 FA 20 00  f+ + + 2+4 40

```

Beispiel (Forts.): ggt-Algorithmus (in Assembler)

```

.....
+0000063D.) 55          PUSH    BP
+0000063E.) 8B EC        MOV     BP,SP
+00000640.) C4 5E 06     LES     BX,SS:[BP+06]
+00000643.) B8 BB 05     MOV     AX,05BB
+00000646.) 33 D2       XOR     DX,DX
+00000648.) E8 EE FE     Relative NEAR CALL [CS:-0112] ; LOCAL LABEL AT +00000539
+0000064B.) 75 0A       SHORT   JNE  [+0A]           ; LOCAL LABEL AT +00000657
+0000064D.) 26          ES:
+0000064E.) 83 7F 1A 00  CMP     DS:[BX+1A],00
+00000652.) 74 03       SHORT   JE   [+03]           ; LOCAL LABEL AT +00000657
+00000654.) E8 70 00     Relative NEAR CALL [CS:+0070] ;LOCAL SUBROUTINE +000006C7
+00000657.) 5D          POP     BP
+00000658.) CA 04 00     FAR     RET 0004
+0000065B.) AC          LODSB
+0000065C.) 3C 0D       CMP     AL,0D
+0000065E.) 74 0C       SHORT   JE   [+0C]           ; LOCAL LABEL AT +0000066C
+00000660.) 3C 1A       CMP     AL,1A
+00000662.) 74 11       SHORT   JE   [+11]           ; LOCAL LABEL AT +00000675
+00000664.) 3B F3       CMP     SI,BX
+00000666.) 75 F3       SHORT   JNE  [-0D]           ; LOCAL LABEL AT +0000065B
+00000668.) B8 BB 05     MOV     AX,05BB
+0000066B.) C3          NEAR    RET
+0000066C.) 3B F3       CMP     SI,BX
+0000066E.) 74 09       SHORT   JE   [+09]           ; LOCAL LABEL AT +00000679
+00000670.) AC          LODSB
+00000671.) 3C 0A       CMP     AL,0A
+00000673.) 74 01       SHORT   JE   [+01]           ; LOCAL LABEL AT +00000676
+00000675.) 4E          DEC     SI
+00000676.) 33 C0       XOR     AX,AX
+00000678.) C3          NEAR    RET
+00000679.) B8 D0 05     MOV     AX,05D0
.....

```

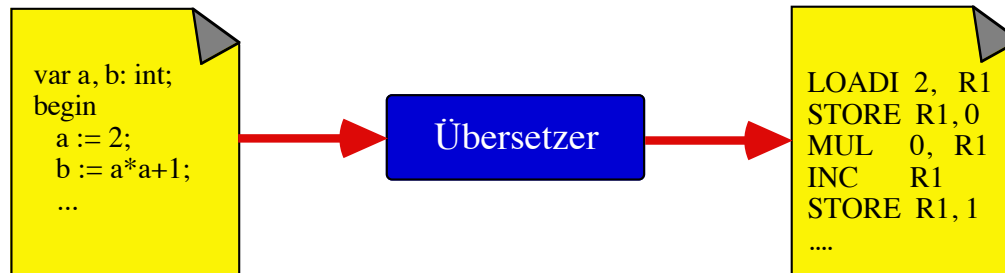

Warum ist Übersetzerbau (auch heute noch) wichtig?

- Etablierte, ausgereifte Teildisziplin der Informatik („Grundlagenwissen“)
- Breiter Anwendungsbereich für Übersetzerbau-Knowhow (z.B. Bau von Software-Werkzeugen)
- Paradebeispiel für erfolgreiches Software Engineering
 - Nahezu vollständige, *theoretische Durchdringung* des Problembereichs und potenzieller Lösungsmöglichkeiten
 - Für jede der Teilaufgaben (eines Übersetzers)
 - problemadäquate Beschreibungsmittel
Formalisierung der Problemstellung
 - Werkzeuge zur automatischen Umsetzung der Beschreibungen in Programme
Generierung von Programmen aus Formalisierungen
 - Konsequente *Strukturierung* (modularer Aufbau von Übersetzern entsprechend Teilaufgaben)

Aufwand 1957: 18 PJ
(für 1. FORTRAN-Compiler)
Aufwand heute: ≤ 1 PJ
(für vergleichbaren Compiler)

Begriffsbildungen (vorläufig)

- **Übersetzer**: Programm zur (semantiktreuen) Umsetzung
(höhere) Programmiersprache $\rightarrow$ Maschinsprache, d.h. Abbildung $\mathbb{U}_L: L \rightarrow M$



- Imperativ: „zustandsorientiert“; Variablen + Anweisungen als wesentliche Konzepte
- Objektorientiert: „zustandsorientiert“; Objekte + Botschaften als wesentliche Konzepte
- Funktional: „wertorientiert“ (nur Ausdrücke);
üblicher Ausführungsmechanismus: **Reduktion**
- Logisch: operationelle Sicht der Prädikatenlogik;
Ausführungsmechanismus: **Resolution**
- Spezielle Sprachen (z.B. Hardwarebeschreibung, Graphik- und Textverarbeitung)

Ersetzung von Ausdrücken durch Werte

*Formel A folgt aus Menge von Formeln P
 $\Leftrightarrow P \cup \{\neg A\}$ ist widerspruchsvoll*

- **Übersetzerbau**: Formalismen, Methoden, Werkzeuge
zur systematischen Erstellung von Übersetzern



Beispiel

$f(x, y) = x + y$
 $f'(y) = 1 + y$
 $f'(y) = f(1, y)$

Basis für Korrektheitsüberlegungen

Universeller Interpreter: $I_u: S \times L \times D^* \rightarrow D^* \cup \{\text{error}\}$

Partielle Auswertung: $I_L = I_u(b_L)$

Sprachspezifischer Interpreter: $I_L: L \times D^* \rightarrow D^* \cup \{\text{error}\}$
Korrektheit: $I_L(p_L, e) = I_u(b_L, p_L, e)$

Interpreter

S	Menge von Sprachdefinitionen
L	Menge der „Quellsprachen“-Programme
M	Menge der „Zielsprachen“-Programme
$b_L \in S$	Beschreibung von L
$b_M \in S$	Beschreibung von M
$p_L \in L$	Quellsprachenprogramm
$p_M \in M$	Maschinenprogramm
D	Datenbereich
$e \in D^*$	Eingabe
error	Sonderelement

Beispiel

$f(x, y) = x^4 + x^2 + y$
 $f'(x) = x^2$,
 $f''(x, y) = x^2 + x + y$
 $f(x, y) = f' \circ f'' = f''(f'(x), y)$

Funktionsaufspaltung: $I_u = \ddot{U}_u \circ I_m$

Universeller Übersetzer: $\ddot{U}_u: S \times L \rightarrow M$
Maschinen-Interpreter: $I_m: M \times D^* \rightarrow D^* \cup \{\text{error}\}$
Korrektheit: $I_m(\ddot{U}_u(b_L, p_L), e) = I_u(b_L, p_L, e)$

Currying: $\ddot{U}_u \cong \ddot{U}_G$

$A \times B \rightarrow C \cong$
 $A \rightarrow (B \rightarrow C)$

Übersetzer-Generator: $\ddot{U}_G: S \rightarrow (L \rightarrow M)$
Korrektheit: $\ddot{U}_G(b_L)(p_L) = \ddot{U}_u(b_L, p_L)$

Auswertung

Sprachspezifischer Übersetzer: $\ddot{U}_L: L \rightarrow M$
Korrektheit: $\ddot{U}_L(p_L) = \ddot{U}_G(b_L)(p_L)$

Insgesamt: $I_m(\ddot{U}_L(p_L), e) = I_u(b_L, p_L, e)$

Übersetzer

Bisher vereinfacht

- Übersetzer = Abbildung: $\ddot{U}_L: L \rightarrow M$

*d.h. $p_L \in L$ als gültiges Programm,
d.h. als syntaktisch und
kontext-korrekt vorausgesetzt*

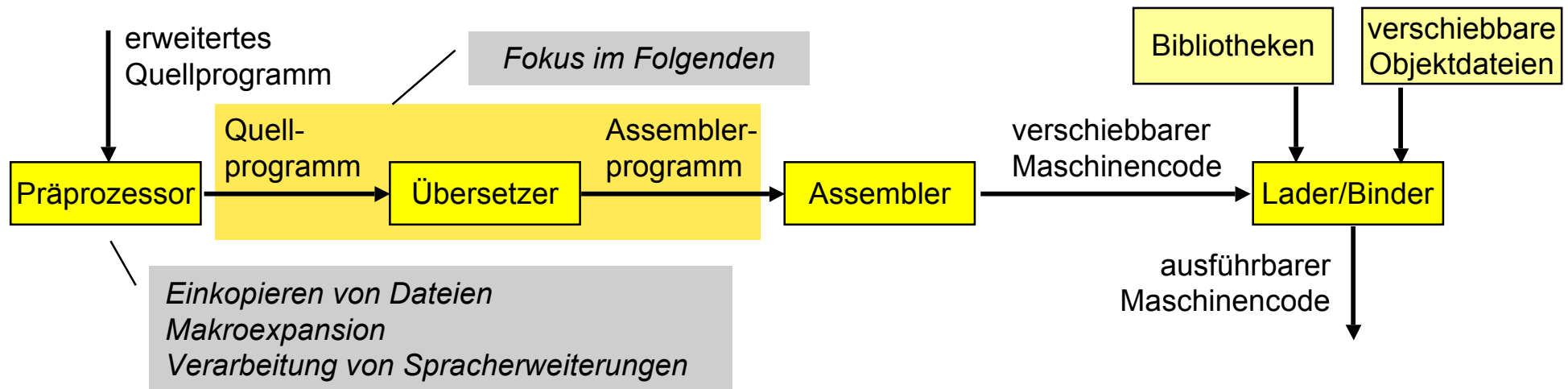
Tatsächlich

- Eingabe: Folge von Zeichen über Eingabezeichenvorrat V
(von der festgestellt werden muss, ob sie ein „gültiges“ Programm der Programmiersprache ist)
- Ausgabe: Folge von Zeichen über einem Ausgabezeichenvorrat W
(die ggf. in bestimmter Weise in der Maschinensprache formatiert sein muss)

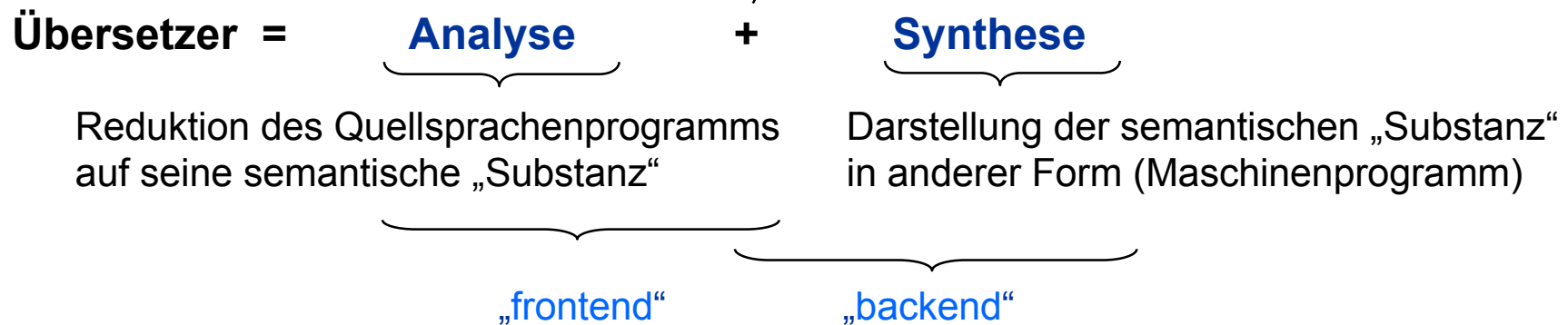
Übersetzer

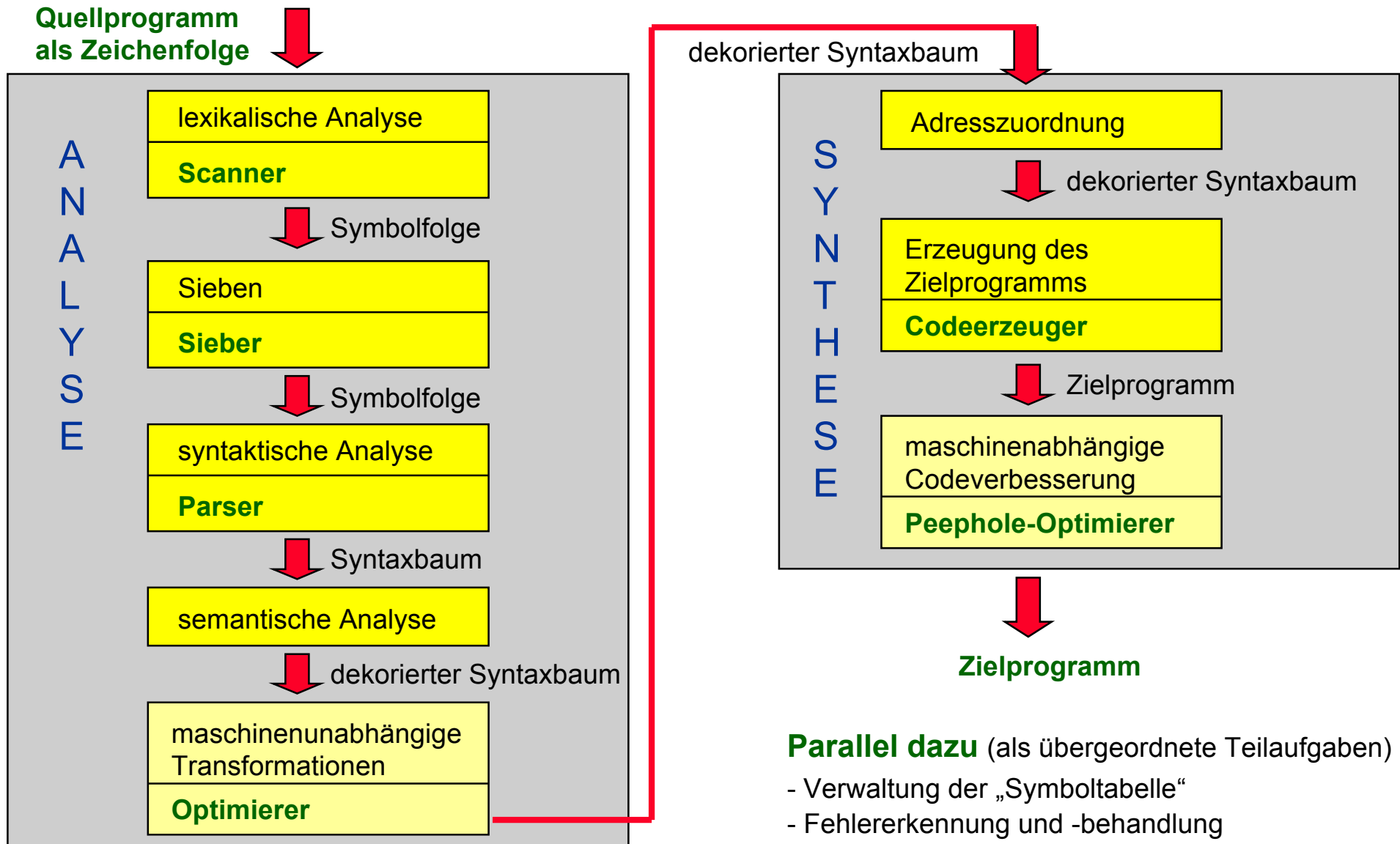
- Komplexe Funktion $\ddot{U}: V^* \rightarrow W^*$
- Bewältigung der Komplexität: Strukturierung durch Funktionsaufspaltung
 $\ddot{U} = \ddot{U}_E \circ \ddot{U}_L \circ \ddot{U}_A$ mit $\ddot{U}_E: V^* \rightarrow L$ und $\ddot{U}_A: M \rightarrow W^*$
 - Geringere Komplexität der Teilfunktionen
 - Individuelle Behandlung der Teilfunktionen

Übersetzer als Bestandteil eines sprachverarbeitenden Systems



Grobstruktur eines Übersetzers





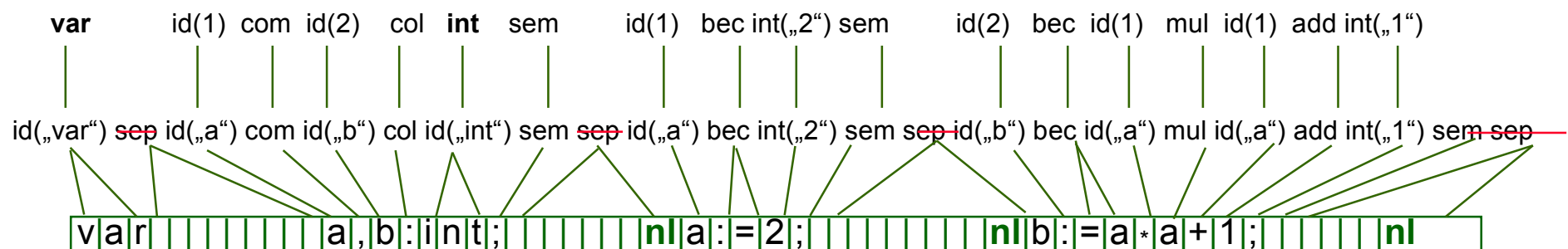
Scanner-Aufgaben

- Quellprogramm als Zeichenfolge lesen
- Zeichenfolge in lexikalische Einheiten („Symbole“) zerlegen, z.B.
 - Bezeichner (id)
 - Zahldenotationen (int)
 - Separatoren (z.B. Zwischenraum, neue Zeile)
 - Sonderzeichen(-Kombinationen) (com, col, sem, bec, mul, add)

Beachte: Symbol ≠ Zeichen

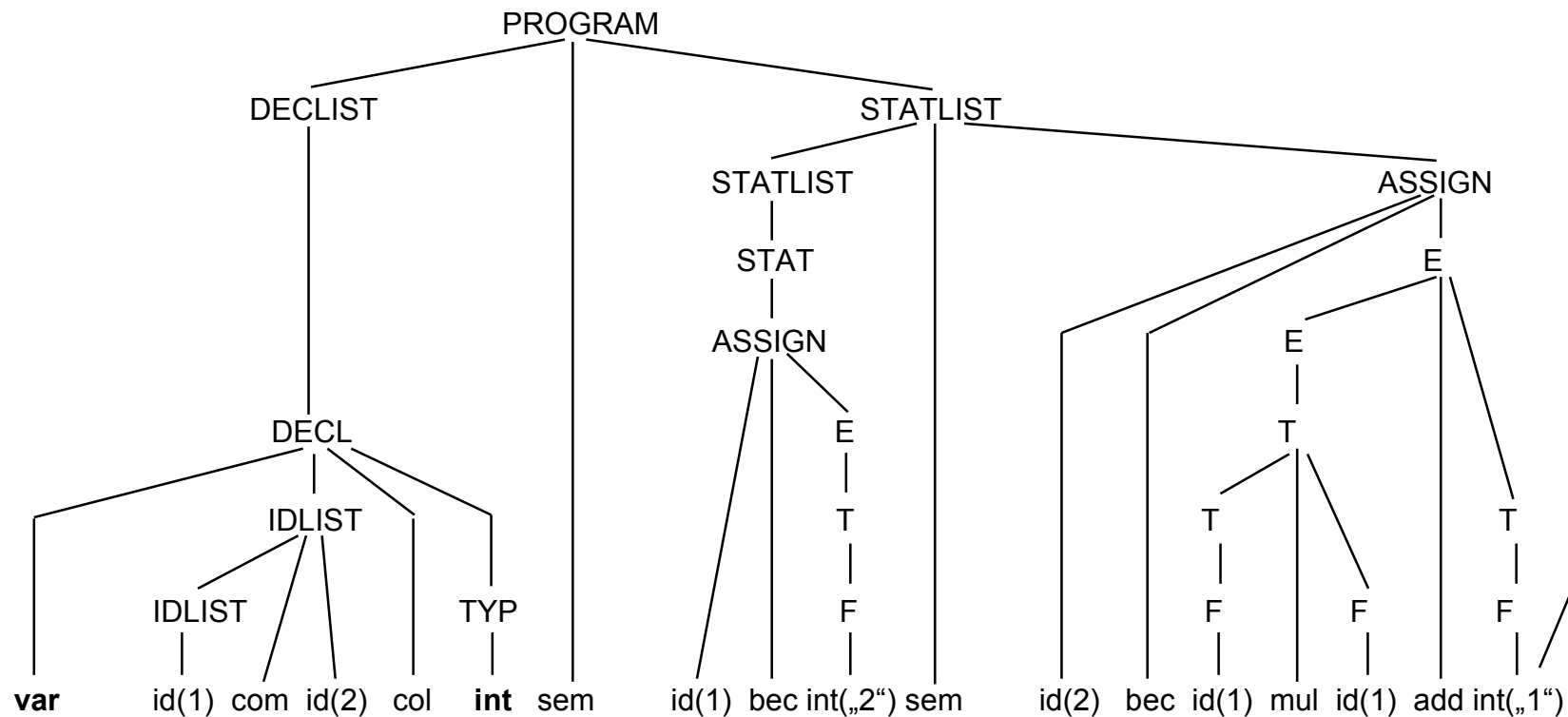
Sieber-Aufgaben

- Besondere Symbole erkennen (und „verarbeiten“), z.B.
 - Übersetzer-Direktiven, Schlüsselwörter
 - Überflüssige Symbole (z.B. Kommentare, Folgen von Leerzeichen)
- Symbole bestimmter Symbolklassen (z.B. id) eindeutig codieren + durch Code ersetzen



Parser-Aufgaben

- *Struktur* des Programms (entsprechend Grammatik) ermitteln
- Gegebenenfalls Fehler erkennen, lokalisieren und diagnostizieren

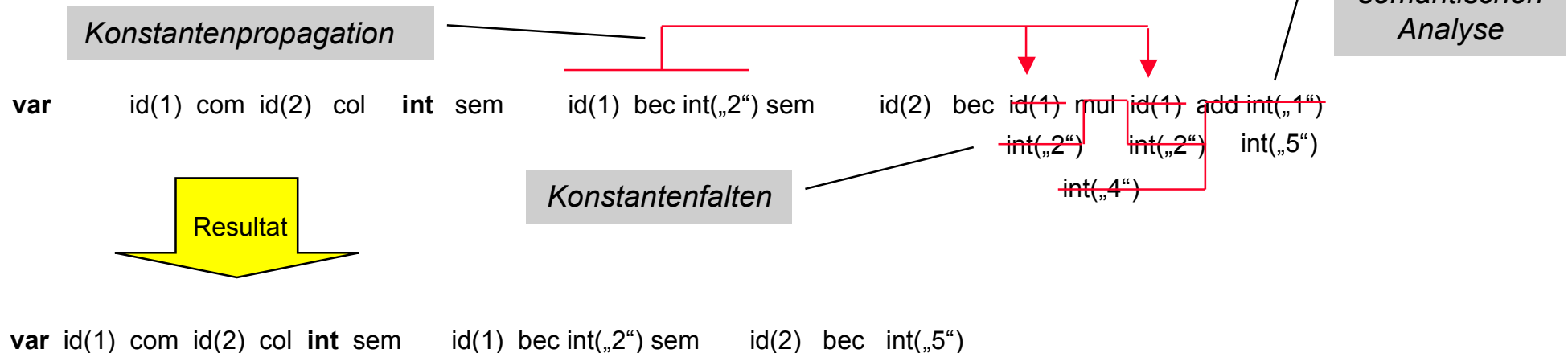


*Resultat
des
Siebers*

- Ermittlung von *statischen semantischen Eigenschaften* ➡
(= Eigenschaften, die über kontextfreie Eigenschaften hinausgehen, aber statisch bestimmt werden können, z.B. *Typisierung* ➡ und *Typkorrektheit* ➡)

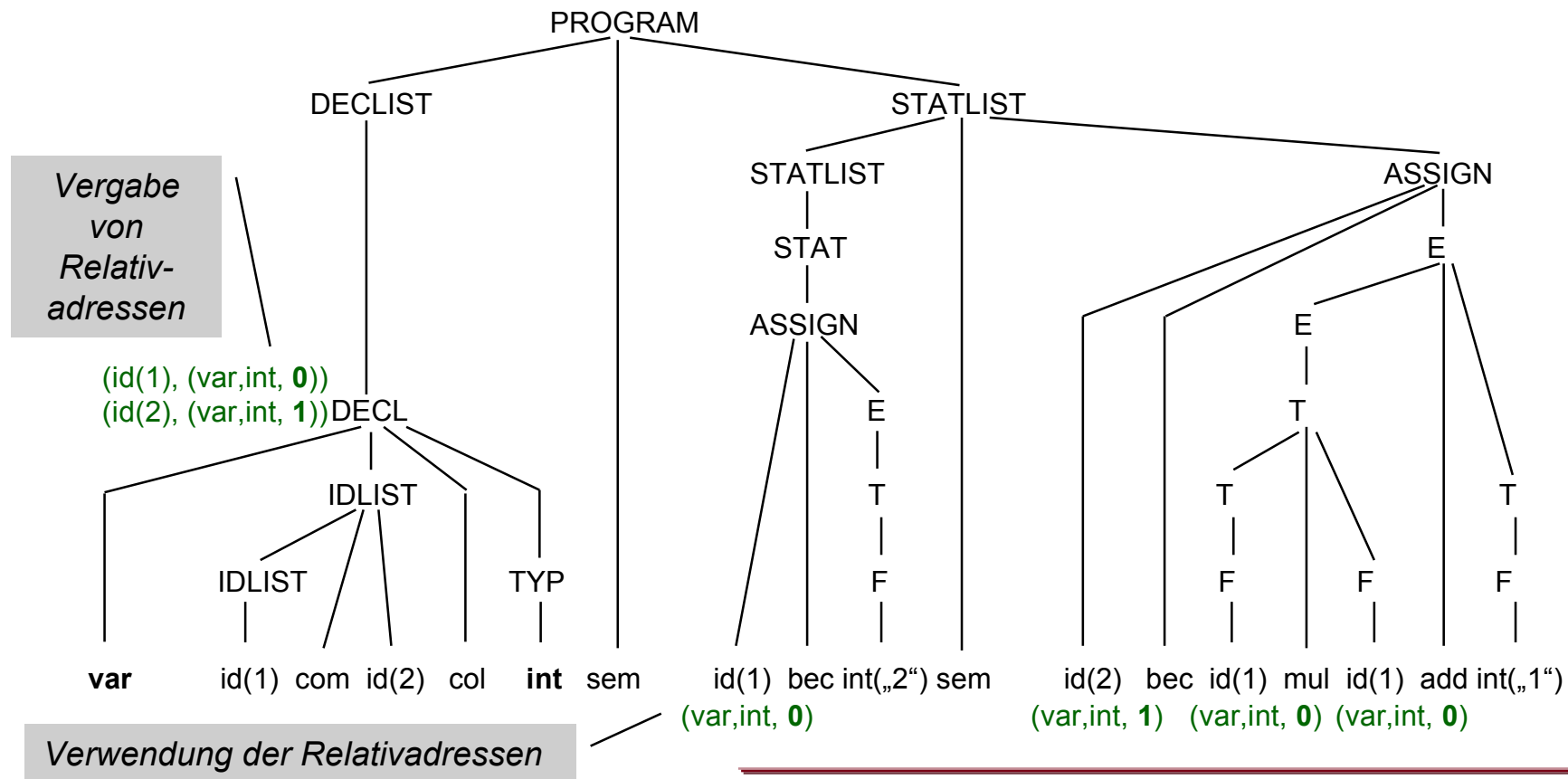
Optimierer-Aufgaben

- Weitere statische Analysen (durch *Datenflussanalyse* oder *abstrakte Interpretation*) hinsichtlich eventueller Laufzeitfehler (z.B. Verwendung uninitialisierter Variablen)
- Effizienzsteigernde *Programmtransformationen* ➡, z.B.
 - Konstantenpropagation
 - Konstantenfalten
 - Herausziehen schleifeninvarianter Berechnungen
 - Elimination von redundanten Berechnungen und „totem Code“



Aufgabe

- Zuordnung von Variablen elementarer Typen zu Speichereinheiten (Relativadressen) unter Verwendung von Eigenschaften der Zielmaschine (z.B. Wort-, Adresslänge, Befehlsvorrat)



Codeerzeuger-Aufgaben

- Generierung der Befehle des Zielprogramms
- *Registerzuteilung* 🖐️ (Kriterium: Effizienz + Ökonomie)
- *Codeselektion* 🖐️ (Auswahl „guter“ Befehlsfolgen)

Resultat der Adressvergabe

var	id(1)	com	id(2)	col	int	sem	id(1) bec int(„2“) sem (var,int, 0)	id(2) bec id(1) mul id(1) add int(„1“) (var,int, 1) (var,int, 0) (var,int, 0)
	LOADI		2,	R1				
	STORE		R1,	0				
	LOAD		0,	R1				
	MUL		0,	R1				
	ADDI		1,	R1				
	STORE		R1,	1				

Maschinenbefehle

LOAD	<i>adr, reg</i>	lade Inhalt von Zelle mit Adresse <i>adr</i> in Register <i>reg</i>
STORE	<i>reg, adr</i>	speichere Inhalt von Register <i>reg</i> in Zelle <i>adr</i>
LOADI	<i>int, reg</i>	lade Konstante <i>int</i> in Register <i>reg</i>
ADDI	<i>int, reg</i>	addiere Konstante <i>int</i> zum Inhalt von Register <i>reg</i>
MUL	<i>adr, reg</i>	multipliziere Inhalt von Register <i>reg</i> mit Inhalt von Zelle mit Adresse <i>adr</i>

Maschinenabhängige Codeverbesserung („Peephole-Optimierung“)

- *Maschinenabhängige* Codeverbesserung braucht nur *lokale Information*
- Maschinenunabhängige Codeverbesserung verwendet meist *globale Information*

Wichtigste Aufgaben

- Überflüssige Befehle eliminieren
- Allgemeine Befehle durch effizientere ersetzen

Resultat der
Codeerzeugung

```
LOADI 2, R1
STORE R1, 0
LOAD 0, R1
MUL 0, R1
ADDI 1, R1
INCR R1
STORE R1, 1
```

Ergebnis: R1 enthält
Inhalt von Adresse 0

Eigenschaften der konzeptionellen Struktur

- Gliederung des Übersetzungsprozesses in eine *Folge von Teilprozessen*
- *Zwischendarstellungen* durch Mechanismen aus der Theorie der formalen Sprachen
- *Aufteilung der Teilprozesse* basiert auf
 - Pragmatischen Überlegungen
 - Korrespondenzen zwischen Beschreibungsmechanismen und Automatenmodellen

In realen Übersetzern

- *Zusammenlegen von Teilaufgaben* (z.B. Scanner + Sieber)
- „*verschränkte*“ *Arbeitsweise* (z.B. Parser ruft Scanner + semantische Routinen auf)
- Vorteil: Aufbau und Speicherung von Zwischenformen kann entfallen

Unterscheidung von Übersetzern nach ihrer (realen) Struktur

- **Einpass-Übersetzer** (nur ein Durchlauf durch das Quellprogramm)
alle Teilaufgaben durch Unterprogramme zum Parser erledigt (z.B. Pascal-Übersetzer)
- **Mehrpas-Übersetzer** (z.B. Algol 68- oder Ada-Übersetzer)
pro Pass eine oder mehrere Teilaufgaben

Formale Spezifikation und Generierung

- (einige) Übersetzerteilaufgaben sind *Erkennungsprobleme* für gewisse Typen von Grammatiken und können durch entsprechende Automaten erkannt werden
 - Grammatiken sind die formale Spezifikation
 - Automaten sind automatisch erzeugbar („Übersetzergenerierung“)

Übersetzerteilaufgaben	Spezifikationsmechanismen	Automatentyp
lexikalische Analyse	reguläre Ausdrücke	determ. endl. Automaten
syntaktische Analyse	kontextfreie Grammatiken	determ. Kellerautomaten
semantische Analyse	Attributgrammatiken	
effizienzsteigernde Transformationen	Baum-zu-Baum-Transformationen	endliche Baumtransduktoren
Codeselektion in der Codeerzeugung	reguläre Baumgrammatiken	endliche Baumautomaten