Grundlagen des Übersetzerbaus (1)

Prof. Dr. Michael Philippsen



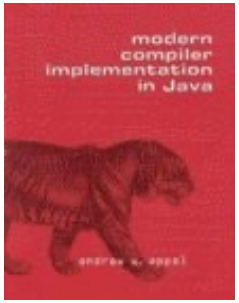




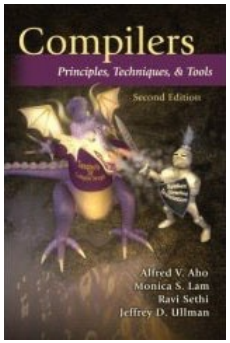

Friedrich-Alexander-Universität
Technische Fakultät

DEPARTMENT INFORMATIK
LEHRSTUHL FÜR PROGRAMMIERSYSTEME
(INFORMATIK 2)

Literatur

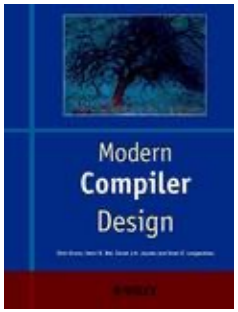


- „Modern Compiler Implementation in Java“, A.W. Appel, Cambridge University Press, 1998



- „Drachenbuch“, A. Aho, R. Sethi, J. Ullmann: Compilers - Principles, Techniques and Tools, Addison-Wesley, 2006

Erste Auflage:



- „Modern Compiler Design“, Grune, Bal, Jacobs, Langendoen, Wiley, 2002

Warum ist Übersetzerbau interessant?

- Erster Übersetzer: Fortran 1954–57
- Stabile Software-Architektur: Musterbeispiel für ein mittelgroßes sequentielles Softwaresystem.
- Hohe Korrektheits- und Zuverlässigkeitsanforderungen.

- Grundlegend für die Weiterentwicklung der Programmiertechnik. Besseres Verständnis für existierende Programmiersprachen. Z.B.:
  - Welche Laufzeitkosten verursacht ein bestimmtes Sprachmerkmal?
  - Wieso nützen finale Variablen/Methoden?
  - Wieso haben nicht alle Sprachen Funktionszeiger?

- Ingenieure können ihre Werkzeuge bauen.

- Standardmethoden für die Verarbeitung textueller Eingabe
- Auch in anderen Kontexten nützlich: Textformatierung (LaTeX), domänenspezifische Sprachen, HTML, XML, Wiki Markup, …

Beispiele zur Verarbeitung textueller Eingaben

- Dokumentbeschreibungssprachen:

```
<LI> Z.B. LaTeX, SGML, HTML. Hier werden Textelemente
wie &Uuml;berschriften, Abs&auml;tze, Aufz&auml;hlun-
gen ebenso wie Darstellungsattribute (<EM>Kursiv-
schrift</EM>) oder Sonderzeichen in Textdateien
beschrieben. </LI>
```

- SQL:

```
select Name, Anschrift
from Studenten
where      Stadt = "Erlangen"
       and Fach = "Informatik"
       and Semester < 10
```

- VLSI-Entwurfssprachen

„übersetzen“

- Text
  - aus Quellsprache L
  - in Zielsprache M
  - zur Ausführung auf einer abstrakten Maschine für M übertragen
  - unter Erhaltung der Bedeutung/Semantik.
- Sprache definiert abstrakte Maschine (und umgekehrt).
- Quell- und Zielsprache können identisch sein (Programmtransformationen).
- Ziel kann eine maschinen-interpretierbare Sprache sein.

Spielarten

- Reiner Interpretierer („Interpreter“)
  - Liest Quelltext jeder Anweisung, interpretiert diese/führt diese aus.
  - Für größere Programme ineffizient.
- Interpretation nach Vorübersetzung
  - Analyse des Quellprogramms und Transformation in eine für den Interpretierer günstigere Form.
  - Beispiele: Java-Bytecode, Smalltalk-Bytecode, Pascal P-Code
- Vollständige Übersetzung → Hauptthema dieser Vorlesung
- Laufzeitübersetzer → „Ausgewählte Kapitel“
  - Erfunden 1974 an der CMU, heute JIT.
  - Auszuführender Code wird übersetzt und eingesetzt:
    - Schneller als reine Interpretation für hinreichend große Programme.
    - Kann dynamisch ermittelte Laufzeiteigenschaften berücksichtigen (dynamische Optimierung).

Randbedingungen der Übersetzung

- Immer: Korrektheit
- Immer: Sicherstellen der programmiersprachlichen Regeln
- Meistens Effizienz:
  - Minimaler Betriebsmittelaufwand zur Laufzeit,
  - hohe Übersetzungsgeschwindigkeit
- Üblich: Zusammenarbeit mit anderen Werkzeugen
  - Assemblierer, Binder
  - Debugger, Analysewerkzeuge
  - Foreign Function Interface zur Anbindung fremder Übersetzer

Formale Semantik

Formale Semantik ist definierbar durch

- Axiomatische Semantik
  - Axiome legen fest, was als gültig angenommen werden kann, nachdem Sprachkonstrukte ausgeführt worden sind.
  - Programme ~ Beweise
  - Beispiel: wp-Kalkül
- Denotationelle Semantik
  - Jedem Sprachkonstrukt wird (rekursiv) eine Bedeutung zugeordnet
  - Programm definiert Abbildung.
- Operationelle Semantik
  - Abstrakte Maschine mit Zuständen wird benutzt.
  - Jedes Sprachkonstrukt ändert Zustände.
  - Programm definiert Zustandsübergänge.

geeignet für Korrektheitsüberlegungen im Übersetzerbau.

Zustände

- Sequentielles Programm durchläuft die Zustände q_0, q_1, q_2, …
  - Zustand q_t besteht aus Objekten, die zum Zeitpunkt t existieren: $q_t = (q_t^0, q_t^1, \ldots, q_t^n)$
  - Operation im Interpretierer entspricht einem Übergang $q_t \rightarrow q_{t+1}$

Zustandsfolge

- Beispiel:

```
assert j > 0;
while (i != j) {
  if (i > j) i = i - j; else j = j - i;
}
System.out.println("ggt=" + i);
```

- Algorithmus durchläuft die Zustandsfolge:

Anfang:	`i=36`	`j=24`
	`i=12`	`j=24`
Ende:	`i=12`	`j=12`
Ausgabe:	`ggt=12`	

Äquivalenter Algorithmus

- Beispiel

```
assert j > 0;
while (j != 0) {
  int tmp = i % j; i = j; j = tmp;
}
System.out.println("ggt=" + i);
```

- Algorithmus durchläuft *andere* Zustandsfolge:

Anfang:	**i=36**	**j=24**
	i=24	**j=12**
Ende:	**i=12**	**j=0**
Ausgabe:	**ggt=12**	

} sind diese Zwischenzustände von außen beobachtbar?

- Einsicht: Algorithmus kann durch besseren (hier: anderen) ersetzt werden, wenn es nur auf das Endergebnis (bzw. nur auf die beobachtbaren Zustände) ankommt.

Beobachtbare Zustände

- Beobachtbar sind:
  - Anfangs- und Endzustand
  - Ein- und Ausgaben
  - Im Allgemeinen ist in einem Zustand q nur ein Teil der darin enthaltenen Objekte beobachtbar.

- Bemerkung:
  - Die Berechnung nicht beobachtbarer Objekte ist unerheblich, wenn sie terminiert.
  - Zwischenzustände sind unerheblich, wenn es nicht unendlich viele sind.

Korrekte Übersetzung – erster Ansatz

Erster Ansatz: Wir betrachten zunächst nur terminierende Programme:

- Bei deterministischen Programmen:
  Folge von beobachtbaren Zuständen.
- Bei nicht-deterministischen Programmen:
  - Azyklischer Graph von beobachtbaren Zuständen
  - Eine Programmausführung entspricht einem Pfad vom Anfangs- zu einem Endzustand im Graphen.

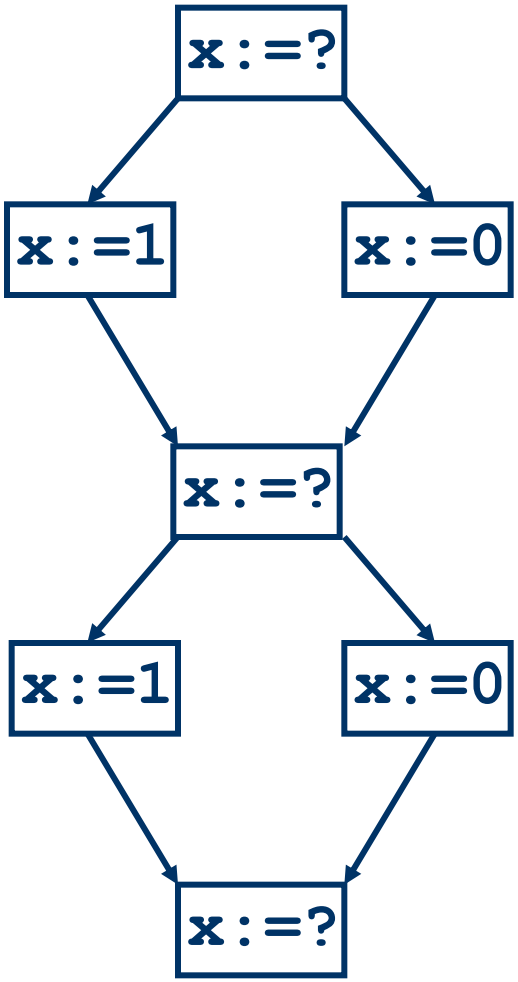
Azyklischer Zustandsgraph

- Indeterministisches Programm

```
do
     true -> x:=1
[]   true -> x:=0
od
```

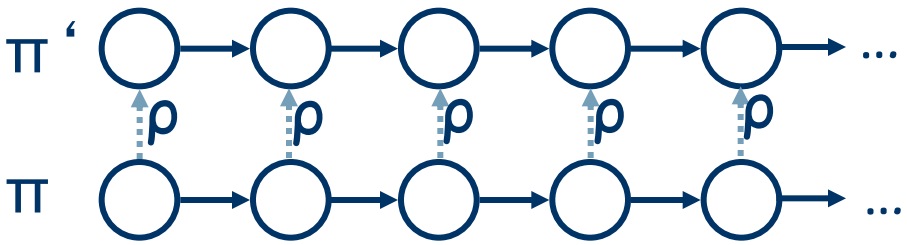
- Konkrete Ausführung

```
do
     true -> x:=1
od
```



Korrekte Übersetzung – unfertig (1)

- Quellprogramm π
- Zielprogramm π ‘
- Abbildung: Bei gleicher Eingabe gleiche beobachtbare Objekte in beobachtbarer Zustandsfolge.
- Relation ρ zwischen beobachtbaren Zuständen:



Eigenschaften korrekter Übersetzung

- Endlichkeit von Betriebsmitteln: Zielprogramm darf mit Fehler enden, auch wenn Quellprogramm „mathematisch“ korrekt ist.
- Beispiel: Weitere Objekt-Allokation, wenn der verfügbare Speicherplatz bereits ausgeschöpft ist.

Korrekte Übersetzung – unfertig (2)

- π ‘ ist korrekte Übersetzung von π, wenn für alle zulässigen Eingaben gilt:
  - Wenn π ‘ regulär (= in endlich vielen Schritte ohne Fehlermeldungen) einen nächsten beobachtbaren Zustand q ‘ erreicht, dann kann π einen Zustand q mit q ρ q ‘ erreichen.
  - Wenn π regulär und deterministisch einen beobachtbaren Zustand q erreicht, dann erreicht π ‘ regulär einen Zustand q ‘ mit q ρ q ‘ oder π ‘ terminiert mit einer Fehlermeldung wegen Verletzung von Betriebsmittelbeschränkungen.
- Achtung:
  - Unscharfe Sprachdefinitionen!
  - Verhalten bei nicht-terminierenden Programmen und bei Programmen, die wegen Betriebsmittelgrenzen abbrechen, offen!

Korrekte Übersetzung – fertig

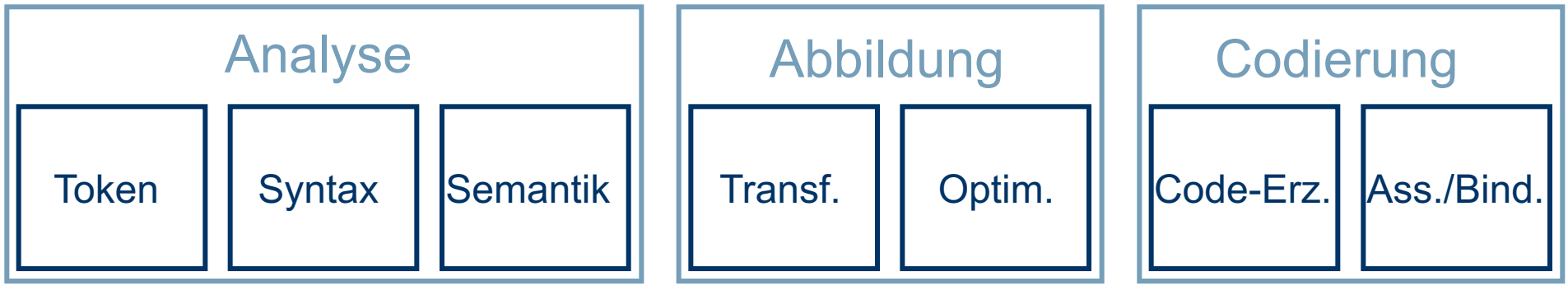
- π ‘ ist korrekte Übersetzung von π, wenn für alle zulässigen Eingaben eine der folgenden Aussagen gilt:
  - Zu jeder Folge beobachtbarer Zustände $q_0´, q_1´, \ldots, q_k´$ von π ‘, die regulär terminiert, gibt es eine terminierende Zustandsfolge $q_0, q_1, \ldots, q_k$ von π mit $q_i \rho q_i´$ für $0 \leq i \leq k$
  - Zu jeder nicht-terminierenden Folge beobachtbarer Zustände $q_0´, q_1´, \ldots$ von π ‘ gibt es eine nicht-terminierende Zustandsfolge $q_0, q_1, \ldots$ von π mit $q_i \rho q_i´$ für alle i.
  - Zu jeder Folge beobachtbarer Zustände $q_0´, q_1´, \ldots, q_{k+1}´$ von π ‘, die mit einer Fehlermeldung wegen Verletzung von Betriebsmittelbeschränkungen terminiert, gibt es eine Zustandsfolge $q_0, q_1, \ldots, q_k, \ldots$ von π mit $q_i \rho q_i´$ für $0 \leq i \leq k$

Korrekte Übersetzung – Unscharfe Sprachdefinitionen

- Beispiel: Überlauf bei Schleifenzähler
  `for (i = v; i <= n; ++i) { … }`
  - Für **`n = maxint`** wird in Fortsetzung irgendwann **`maxint+1`** berechnet werden.
  - In Java: Für **`n = maxint`** terminiert Schleife nicht wegen Überlauf (**`maxint+1 = -maxint-1`**).
  - In C: Verhalten nicht definiert, Schleife terminiert möglicherweise.
- Relation ρ muss auch undefiniertes Verhalten berücksichtigen.

Komponenten des Übersetzers

- Konzeptionelle Struktur:
  - Analysephase
  - Abbildungsphase
  - Codierungsphase
  - Fehlerbehandlung
- Je nach zu übersetzender Programmiersprache können sich die Phasen zeitlich überlappen.



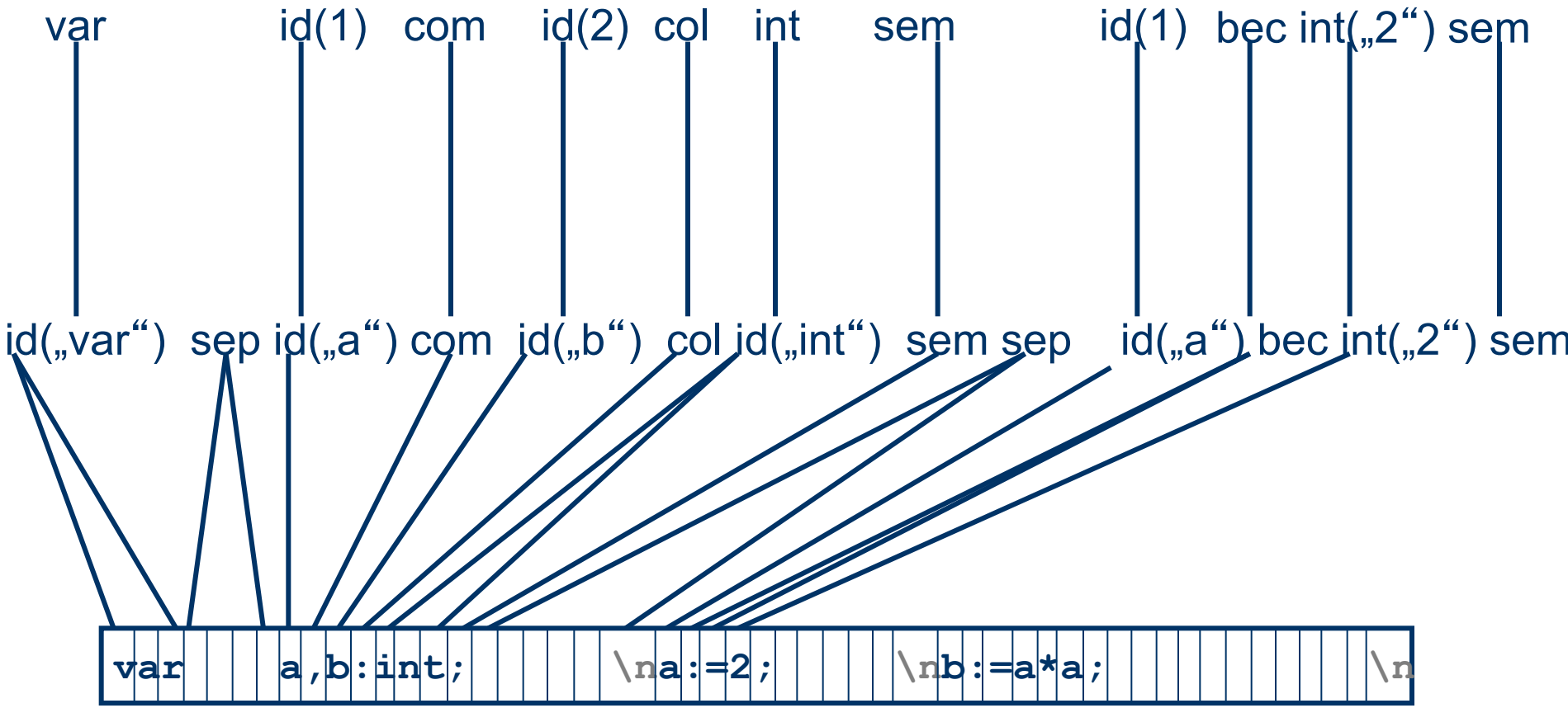
Analysephase

- Aufgaben:
  - Feststellung der bedeutungstragenden Elemente
  - Zuordnung statischer Bedeutung
  - Konsistenzprüfung
- Schritte:
  - Lexikalische Analyse/Abtastung (Lexer)
  - Syntaktische Analyse/Zerteilung (Parser)
  - Semantische Analyse
- Noch keine Übersetzung, nur Analyse

Lexikalische Analyse/Abtastung

- Ein lexikalischer Abtaster/Symbolentschlüsseler (Lexer) zerlegt das
  - Quellprogramm (Text, Folge von Zeichen) in eine Folge von bedeutungstragenden Einheiten/Symbolen (Token),
  - beseitigt überflüssige Zeichen(folgen) wie
    - Kommentare,
    - Leerzeichen, Tabulatoren, …
- Deterministischer endlicher Automat
- Es gibt Werkzeuge zur Generierung effizienter Abtaster aus Spezifikationen.

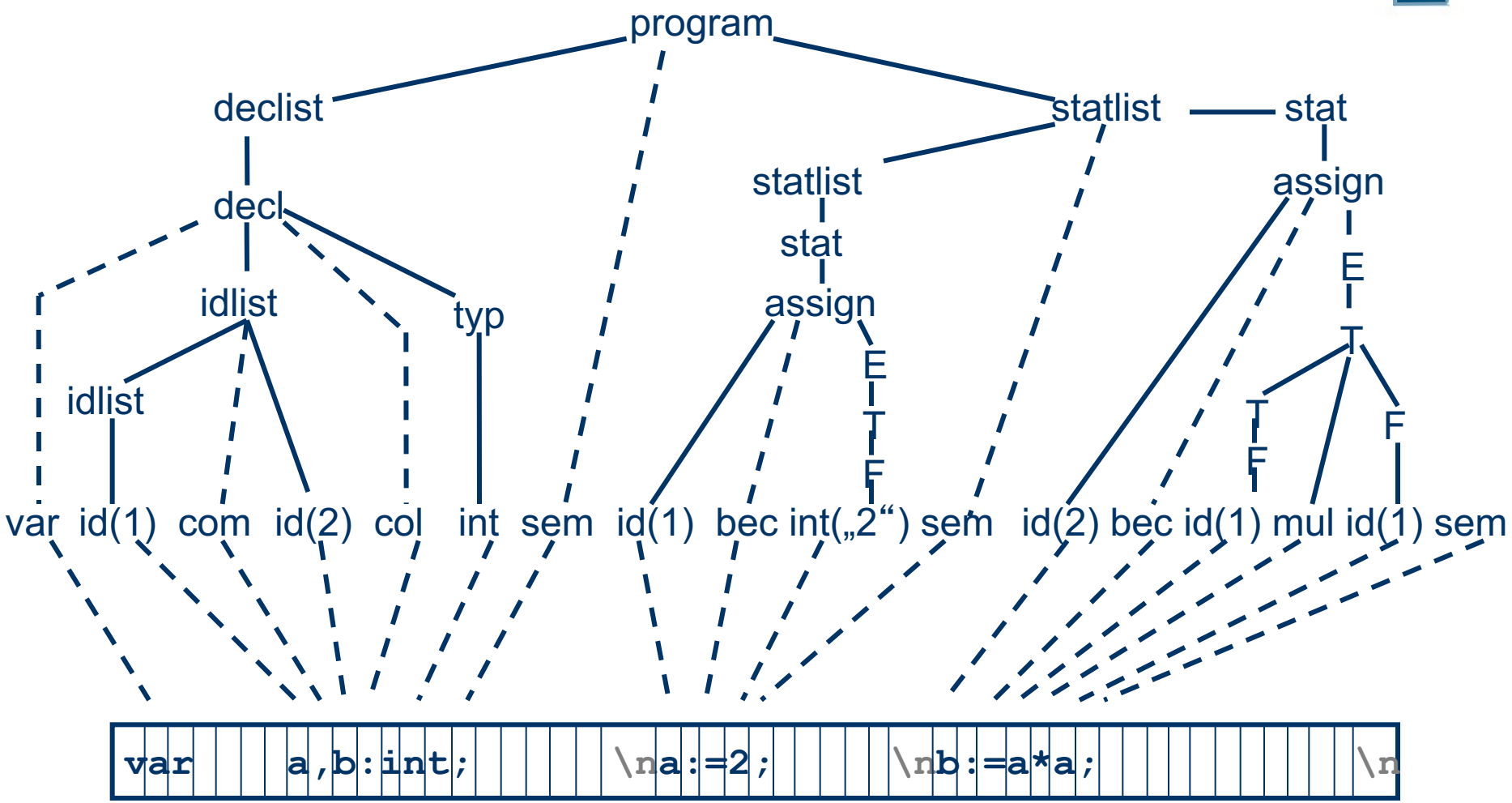
Abtastungsbeispiel



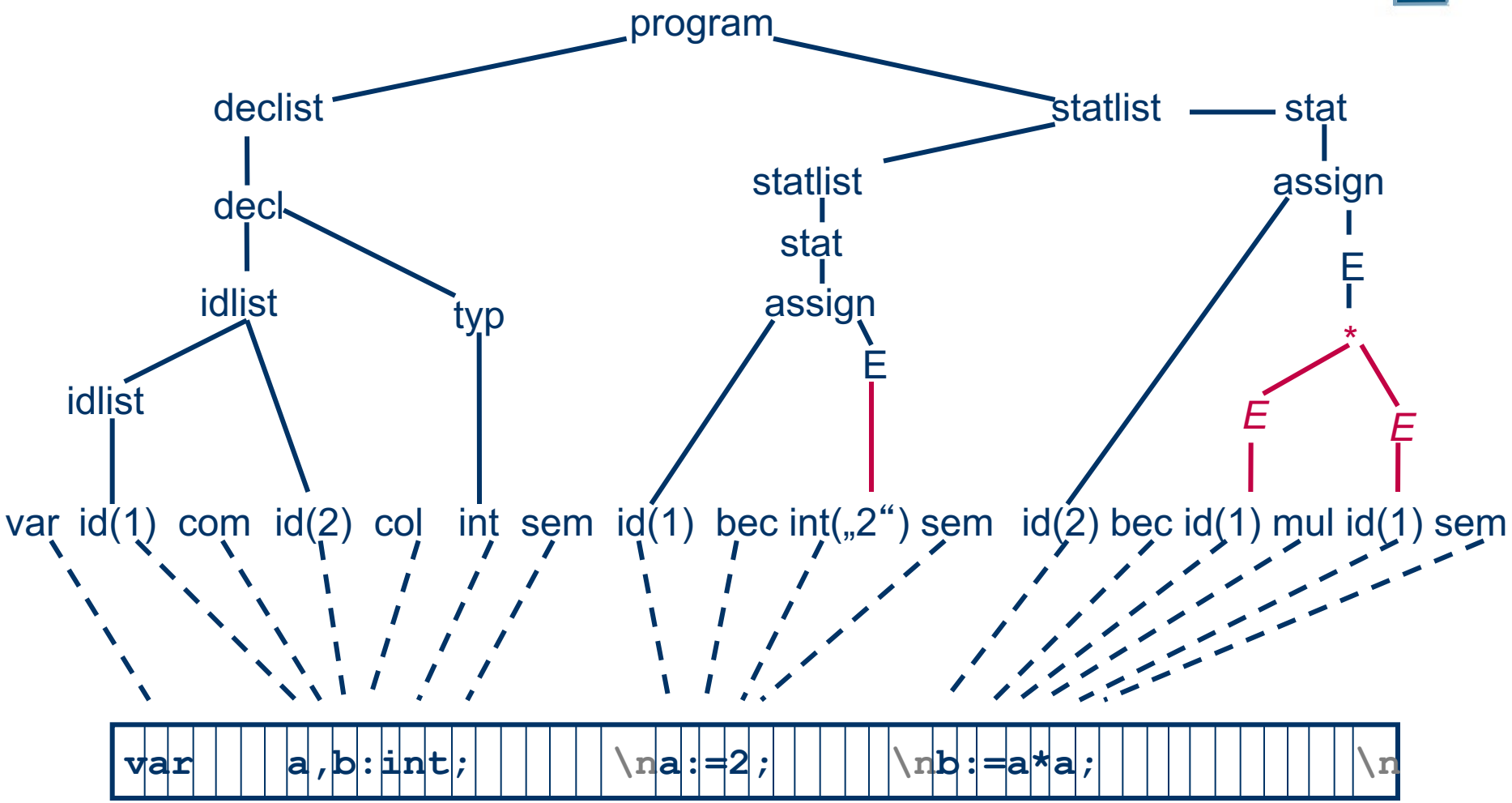
Syntaktische Analyse/Zerteilung

- Die syntaktische Analyse
  - findet die über die lexikalische Struktur hinausgehende Struktur des Programms,
  - kennt Aufbau von Ausdrücken, Anweisungen, Deklarationen, …
- Ein Zerteiler (Parser)
  - transformiert eine Tokenfolge in einen Syntaxbaum/Strukturbaum des Programms
  - gemäß der Grammatik der Sprache (Syntax) und
  - erkennt, lokalisiert und diagnostiziert Fehler.
- Es gibt Werkzeuge zur Generierung effizienter Zerteiler aus Spezifikationen.

Zerteilungsbeispiel: konkrete Syntax



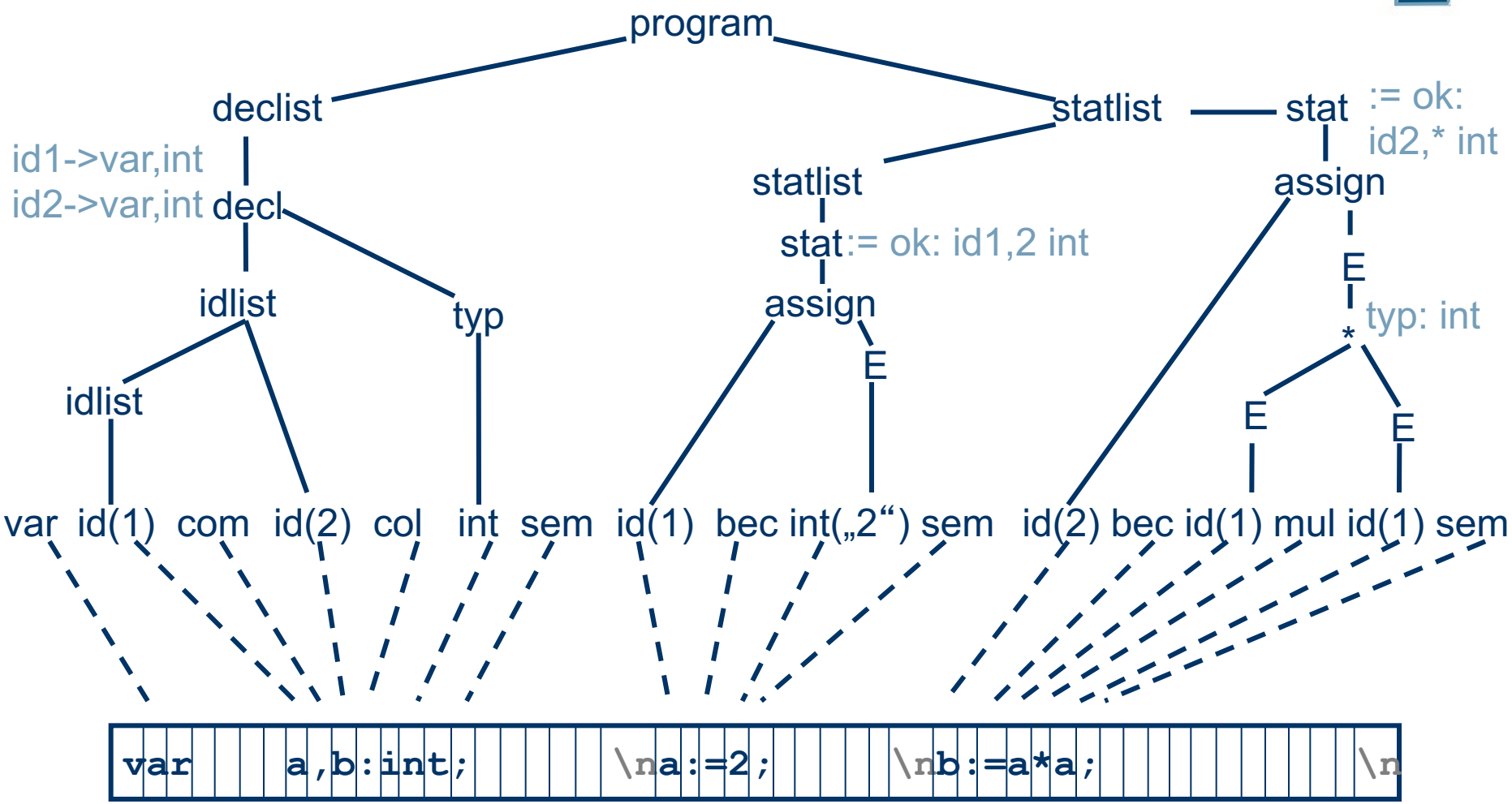
Zerteilungsbeispiel: abstrakte Syntax



Semantische Analyse

- Analyse des Strukturbaums
  - Notwendig, da Programmiersprachen-Regeln kontextsensitiv
  - Beispiel: Definitionen von Bezeichnern ermitteln
- Namens- und Typanalyse
- Konsistenzprüfungen
- Bedeutungsbindung (soweit möglich)
  - Operatoridentifikation
  - Zuordnung von Namensverwendung zu ihrer Definition
- Semantische Analyse wird üblicherweise ad-hoc umgesetzt.

Beispiel für semantische Analyse

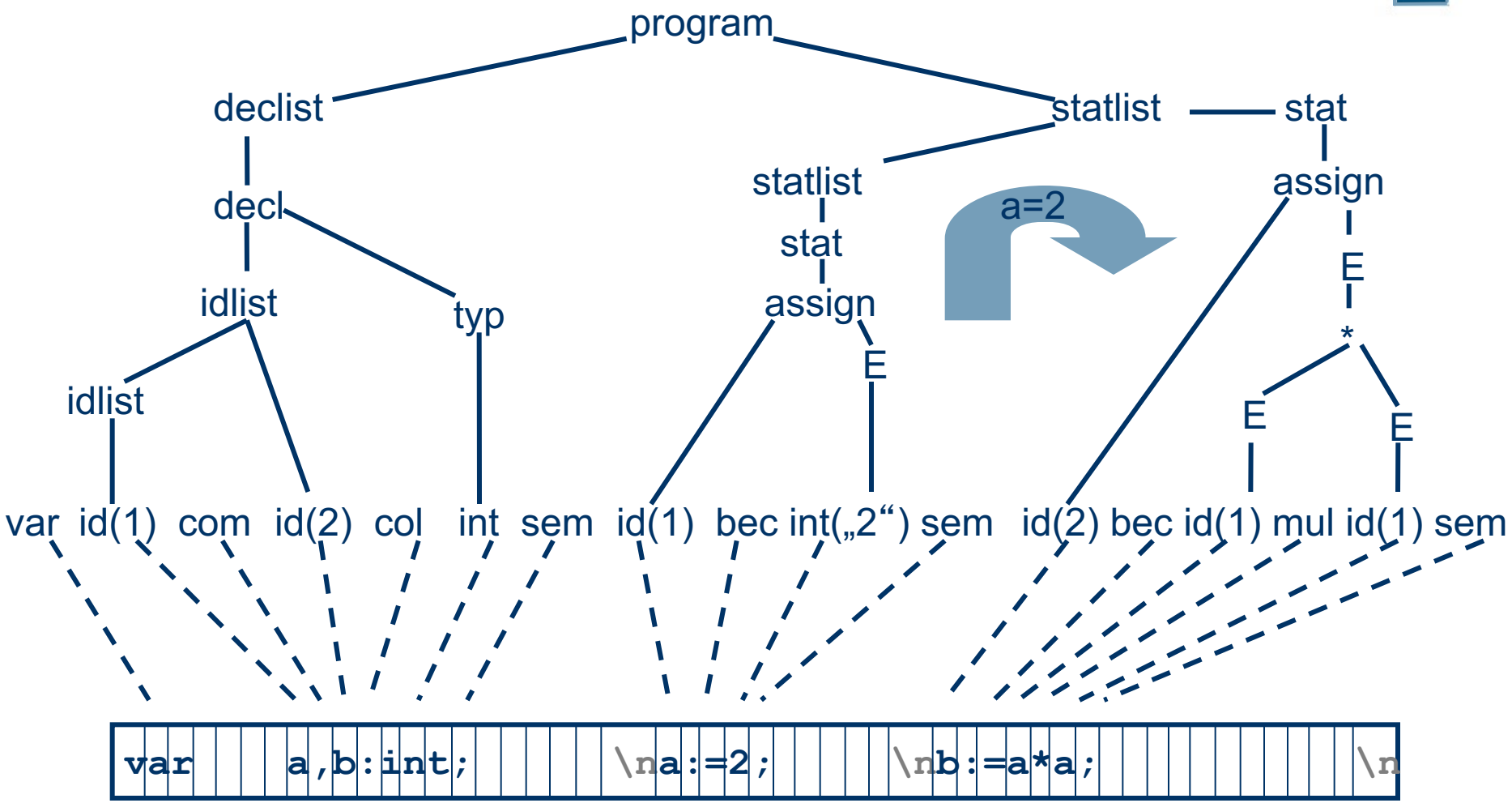


Abbildungsphase

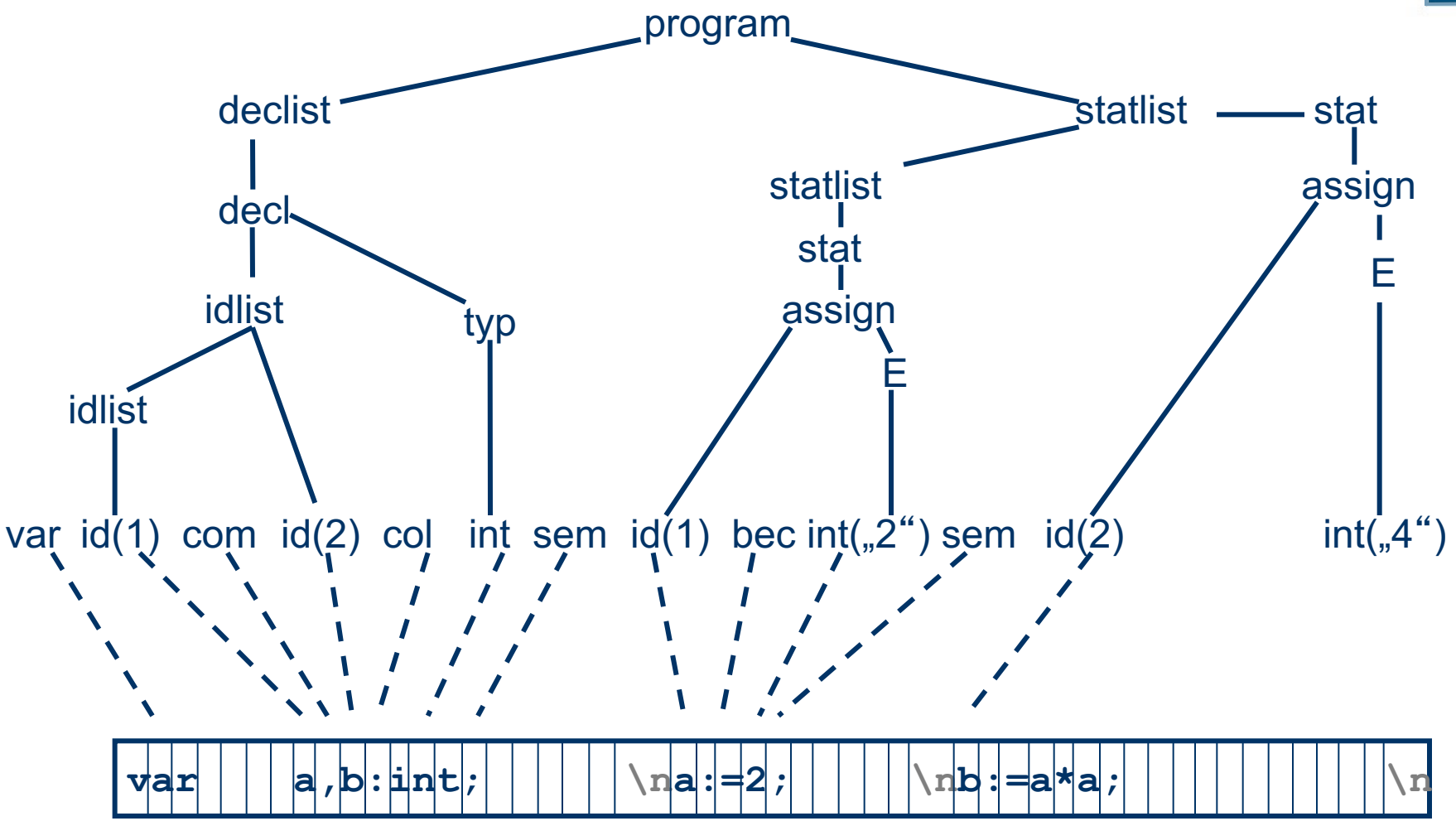
- Programmtransformationen
- Eventuell optimierende Transformationen (mit Nachweis der Anwendbarkeit) → Hauptthema der Vorlesung „Optimierungen im Übersetzerbau“

- Eigentliche Übersetzung
  - Übersetzung der Operationen
  - Übersetzung der Ablaufsteuerung
  - …
- Aber noch keine Generierung des Ziel-Codes
- Üblicherweise nimmt man unbeschränkte Betriebsmittel an

Beispiel für optimierende Transformationen (1)



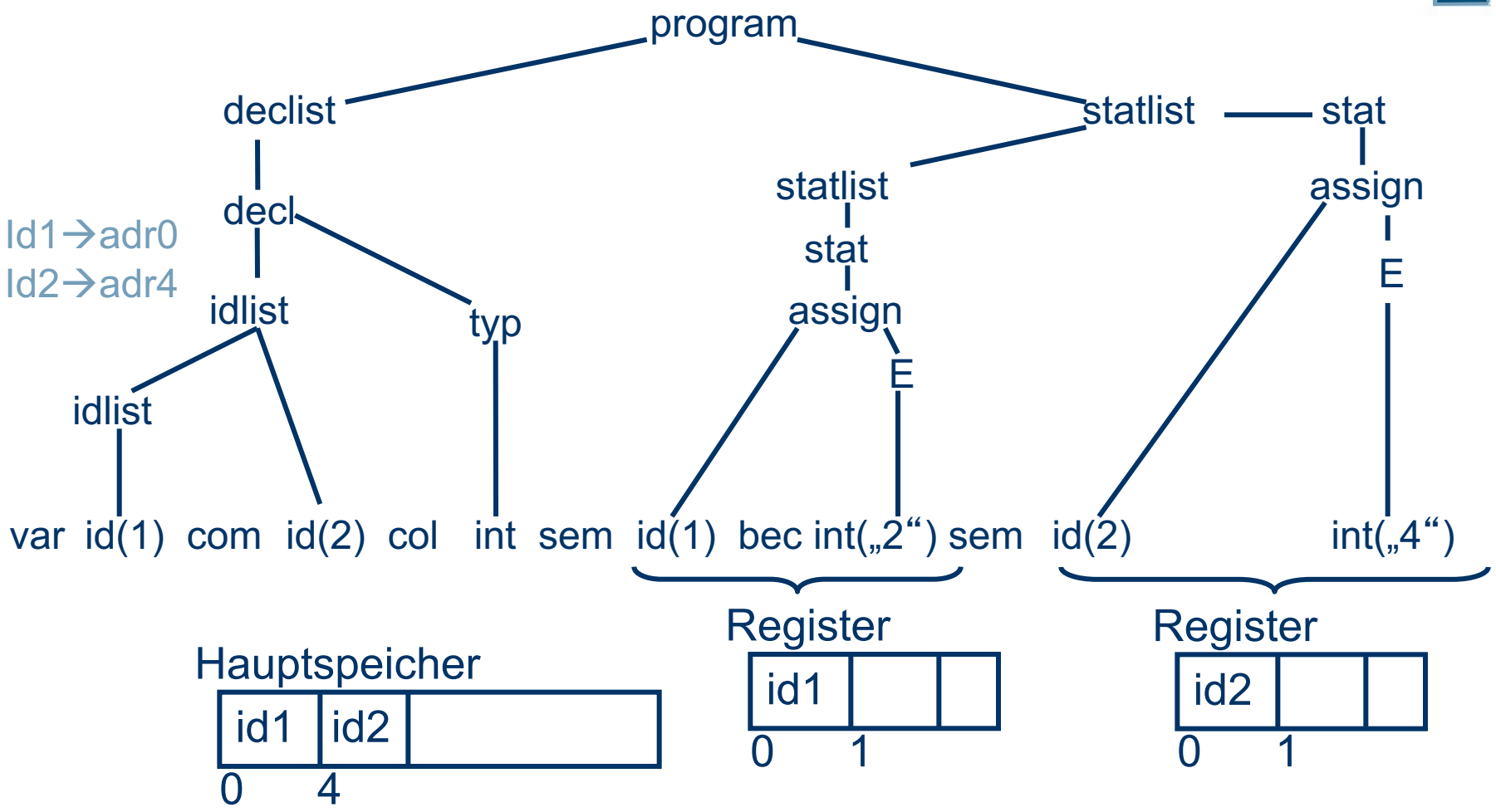
Beispiel für optimierende Transformationen (2)



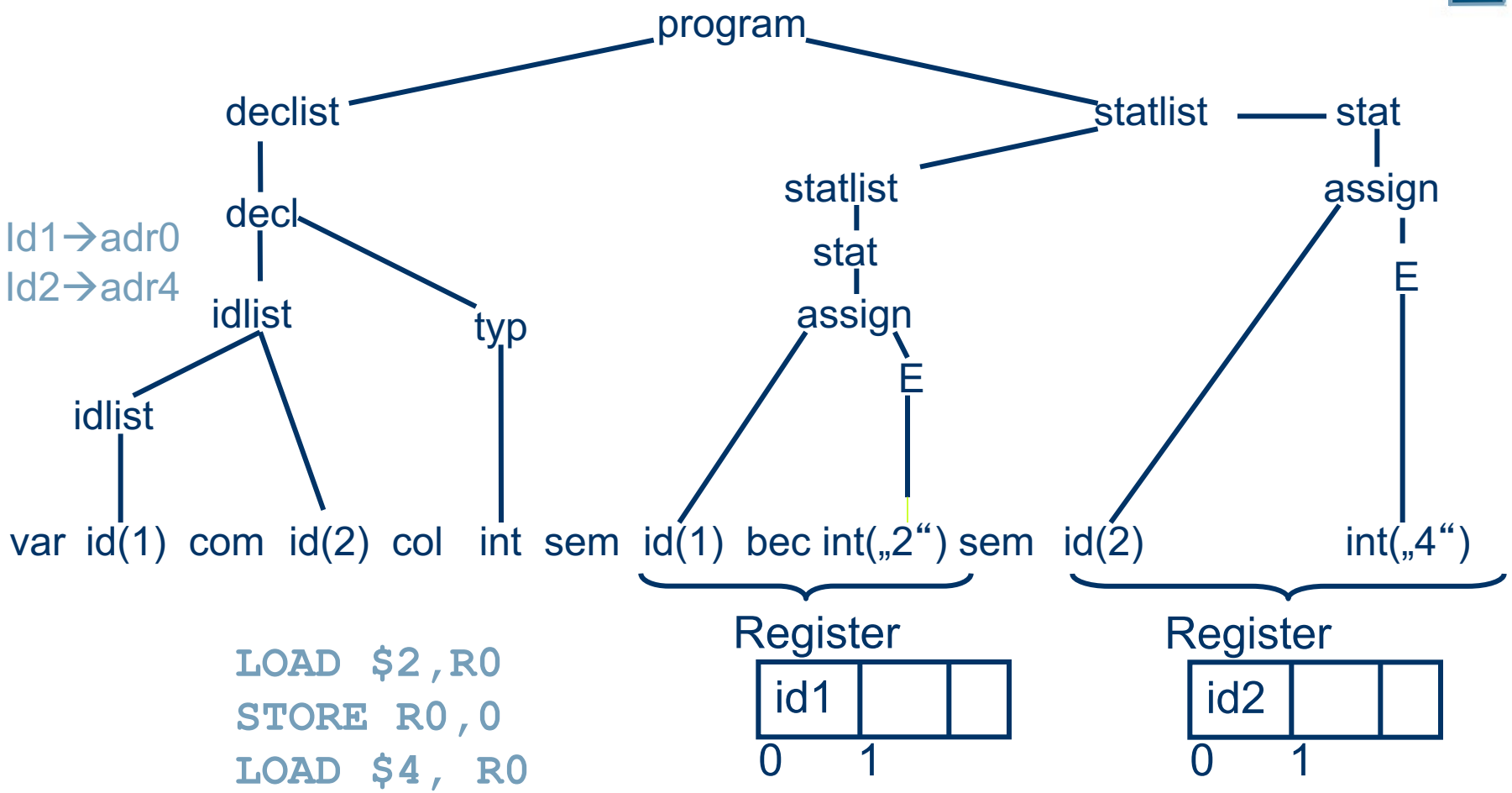
Codierungsphase

- Codierung des Programms in der Zielsprache…
- … unter Berücksichtigung der beschränkten Betriebsmittel der Zielarchitektur
  - Codierung der Instruktionen im entspr. Befehlssatz
  - Abbildung auf Speicherlayout

Beispiel: Speicherlayout, Register



Beispiel: Erzeugung von Code

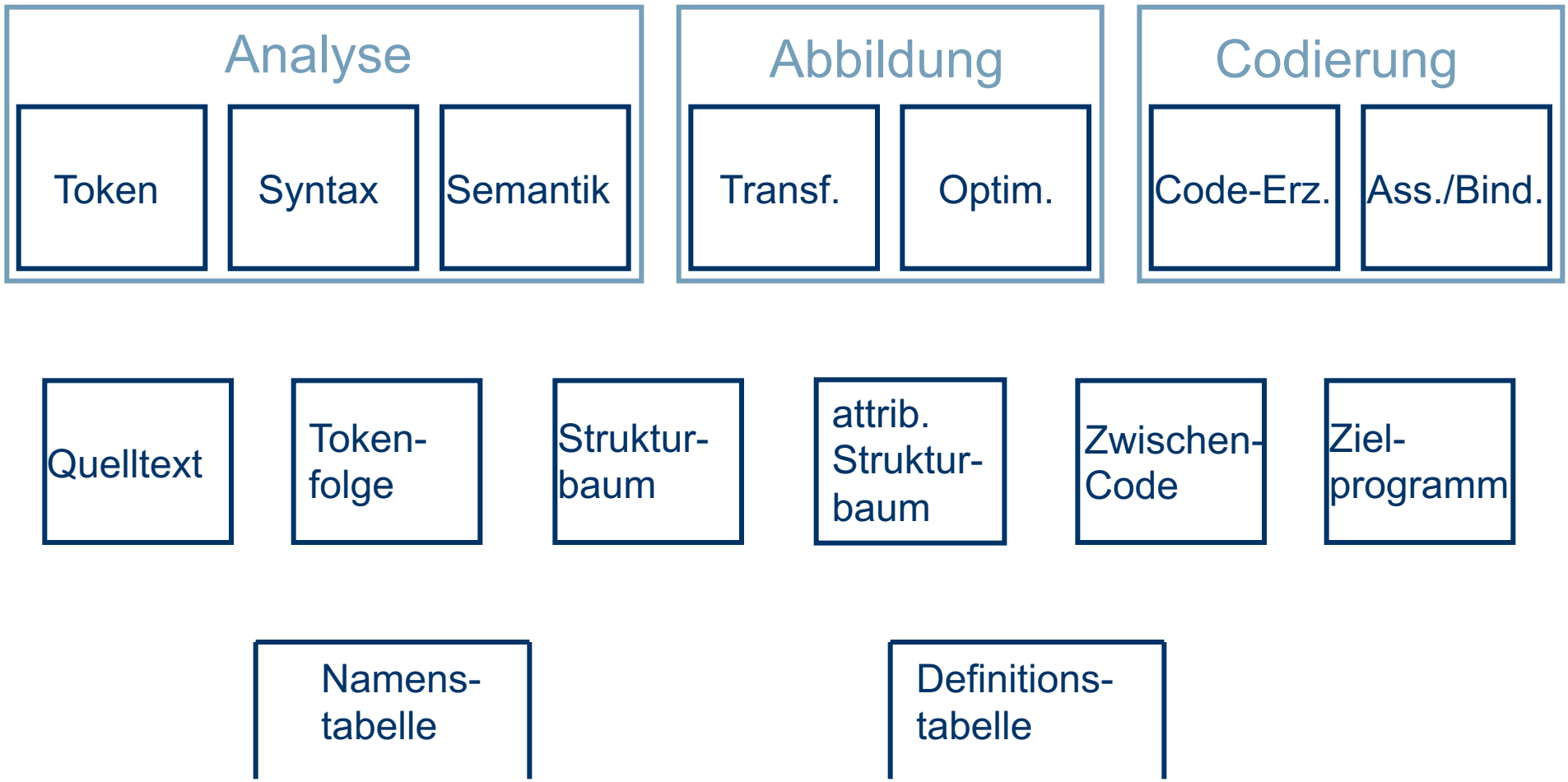



```
LOAD $2,R0
STORE R0,0
LOAD $4, R0
STORE R0,4
```

Aufgaben eines Übersetzers

Analyse	Abtastung	Lesen, Token erkennen
	Syntaktische Analyse (Zerteilung)	Strukturbaum erzeugen
	Semantische Analyse	Namens-, Typ-, Operationen-analyse, Konsistenzprüfung
Abbildung	Transformation	Daten + Operationen abbilden
	Optimierung	Konstantenfaltung, gem. Teil-ausdrücke, globale Zus.hänge erkennen und nutzen
Codierung	Code-Erzeugung	Ausführungsreihenf. festlegen Befehlsauswahl, Register
	Assemblieren u. Binden	Interne und externe Adressen auflösen, Befehle codieren

Modulare Struktur von Übersetzern



Tokenfolge

- Token: Bedeutungstragende Einheit im Programm
- Tokenfolge: Darstellung des Quellprogramms als Folge von Token
- Schnittstelle zwischen Abtaster und Zerteiler

Attributierter Strukturbaum, AST

- Strukturbaum:
  - Schachtelung der bedeutungstragenden Einheiten.
- Abstrakte Syntax:
  - Z.B. Schlüsselworte wie `var` oder `while`, Satzzeichen und Klammerungen werden in der Struktur codiert und sind daher für den Strukturbaum unerheblich.
- Attribute:
  - Ergebnisse der semantischen Analyse.
  - Z.B. Typen, Namen, (konstante) Werte, Definitionsstellen, …
- Schnittstelle zwischen Syntaxanalyse, semantischer Analyse, Abbildungsphase.

Zwischen-Code

- Darstellung des Programms auf niedrigerem Abstraktionsniveau.
  - Näher an Zielsprache.
  - Z.B. keine expliziten Schleifen, keine zusammengesetzten Ausdrücke.
- Befehl für Operationen noch nicht ausgewählt (Art des Registerzugriffs, Adressierungsmodi, … sind noch offen).
- Keine Register- und Speichereinschränkungen.
- Schnittstelle:
  - Transformationsphase und Code-Generierung.
  - Eingabe und Ergebnis vieler Optimierungen.

Zielprogramm

- Ausgabe der Code-Selektion: symbolisch codiert
- Assemblierer (Assembler): codiert binär, löst symbolische Adressen auf, soweit intern bekannt.
- Binder: löst externe Symbole auf (soweit bekannt)
- Ergebnis: Objektprogramm ohne symbolische Adressen.

Namenstabelle

- Zuordnung Bezeichner, Literalkonstanten (evtl. andere Zeichenfolgen aus dem Quellprogramm) zur übersetzer-internen Codierung.
- Aufbau durch Abtaster, dann unverändert im Übersetzungslauf.

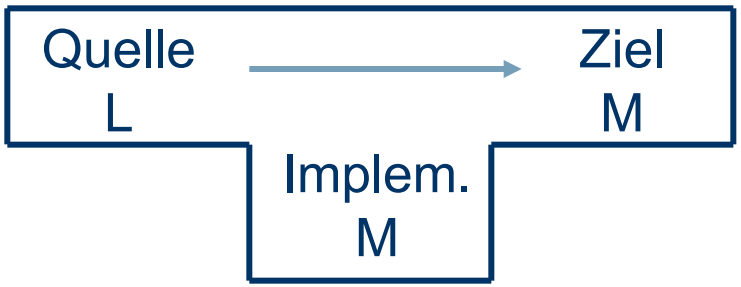
Symboltabelle/Definitionstabelle

- „Datenbank“ des Übersetzers
- Tabelle aller Definitionen (Vereinbarungen):
  - Zuordnung von Token/AST-Knoten zu Bedeutungen.
  - Wann bezeichnen unterschiedliche Token/AST-Knoten dieselben Objekte, Prozeduren?
  - Speichert den analysierten semantischen Kontext von AST-Knoten.

Bootstrapping („sich an den Haaren aus dem Sumpf ziehen“)

- Gegeben:
  - Quellsprache L
  - Abstrakte Maschine M
- Gesucht:

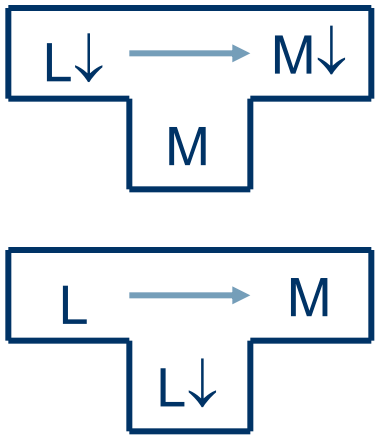
Tombstone-Diagramm



Schwierig. Eigentlich will man den Übersetzer lieber schon in L (anstatt in der Implementierungssprache M) implementieren. Dazu braucht man aber bereits einen Übersetzer für L…

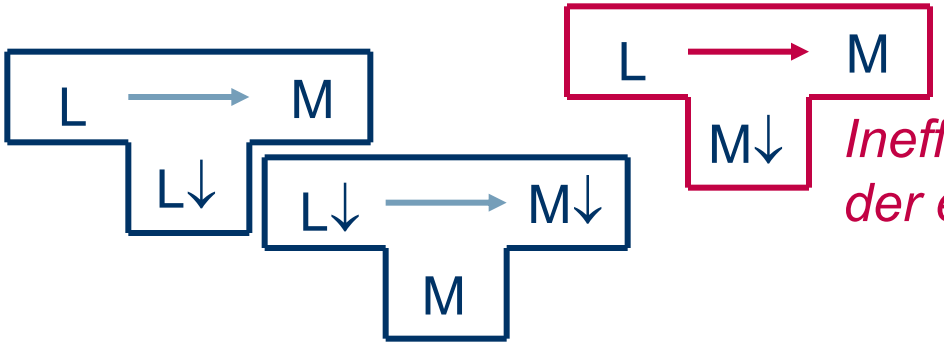
Bootstrapping (2)

- Man definiert sich daher L↓, eine einfache Untermenge von L.
- Man wählt M↓, eine einfache/ineffiziente Untermenge von M aus.
- Man implementiert zwei Übersetzer:



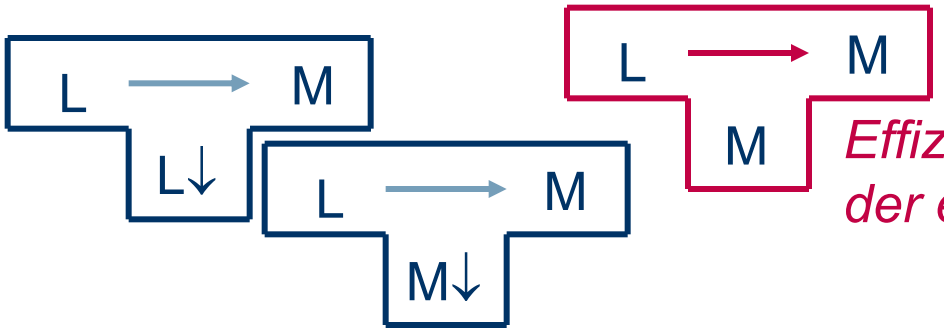
Bootstrapping (3)

- Durch Kombination erhält man:



Ineffizienter Übersetzer, der effizienten Code generiert.

- Durch erneute Kombination erhält man:



Effizienter Übersetzer, der effizienten Code generiert.