Compilerbau für die Common Language Run-Time

Syntax und Semantik von Programmiersprachen

© 2006 Martin v. Löwis

Compilerbau

- Sprachbeschreibung vs. Implementierung
- Beschreibung: formale oder informale (engl.) Lexik, Syntax, statische Semantik, dynamische Semantik
- Implementierung: Algorithmus in einer Programmiersprache
  - etwa: Compiler (Lexik, Syntax, statische Semantik), Laufzeitsystem (dynamische Semantik)
- Compilerwerkzeuge: automatische/halbautomatische Überführung der Sprachbeschreibung in einen Algorithmus

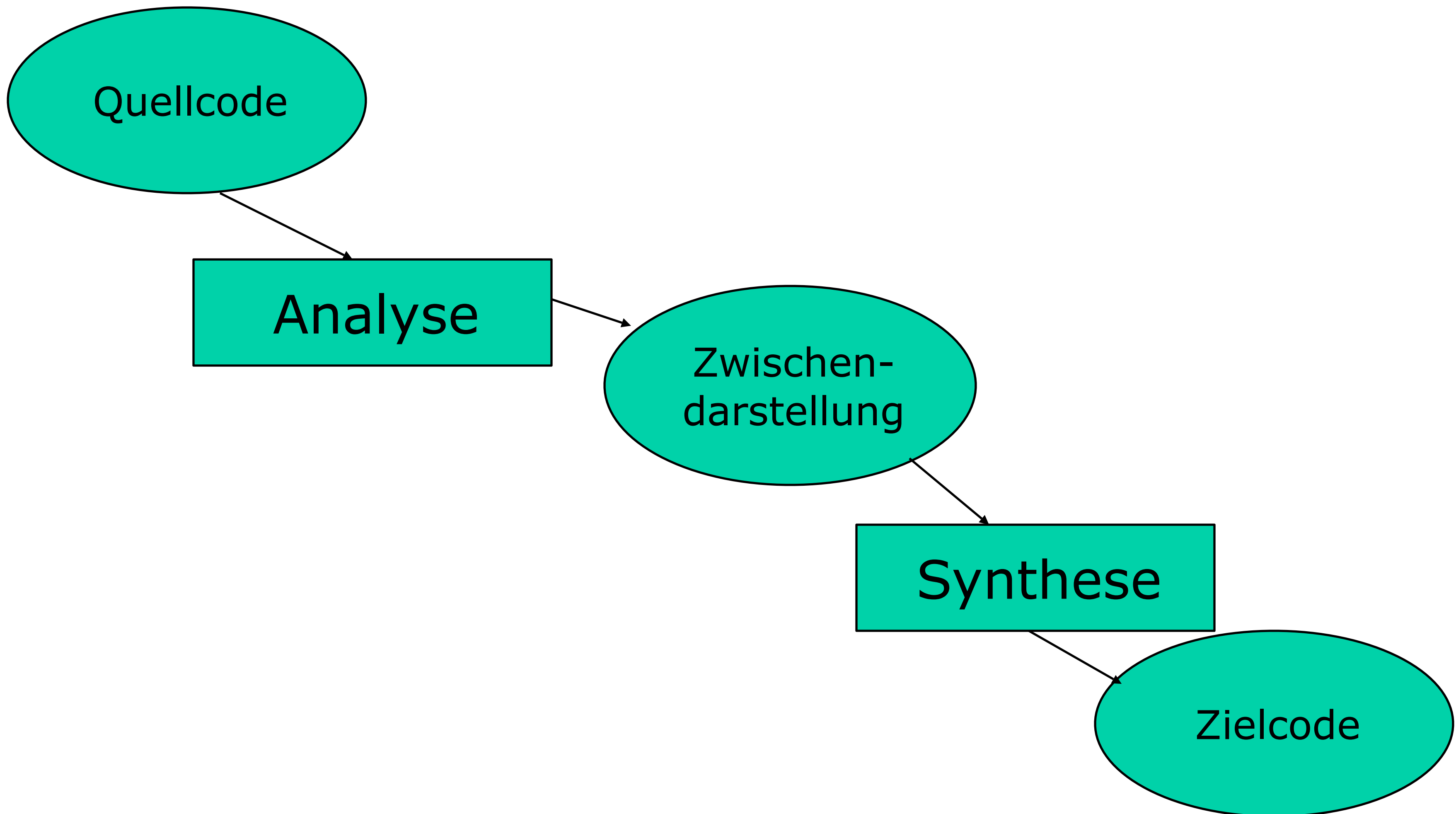
© 2006 Martin v. Löwis

Die As-If-Regel

- “Eine Implementierung einer Programmiersprache darf jede beliebige Strategie wählen, solange das nach außen sichtbare Verhalten so ist, **als ob** die Strategie der Sprachbeschreibung gewählt worden wäre.”
- Nach außen sichtbares Verhalten
  - C: Aufrufe von Ein-/Ausgabefunktionen, Schreib/Lese-Operationen von *volatile*-Variablen
  - Occam: ???
    - vielleicht: Ein-/Ausgabe auf Kanälen zur Umgebung
    - Schreiboperationen auf ausgewählte Variablen

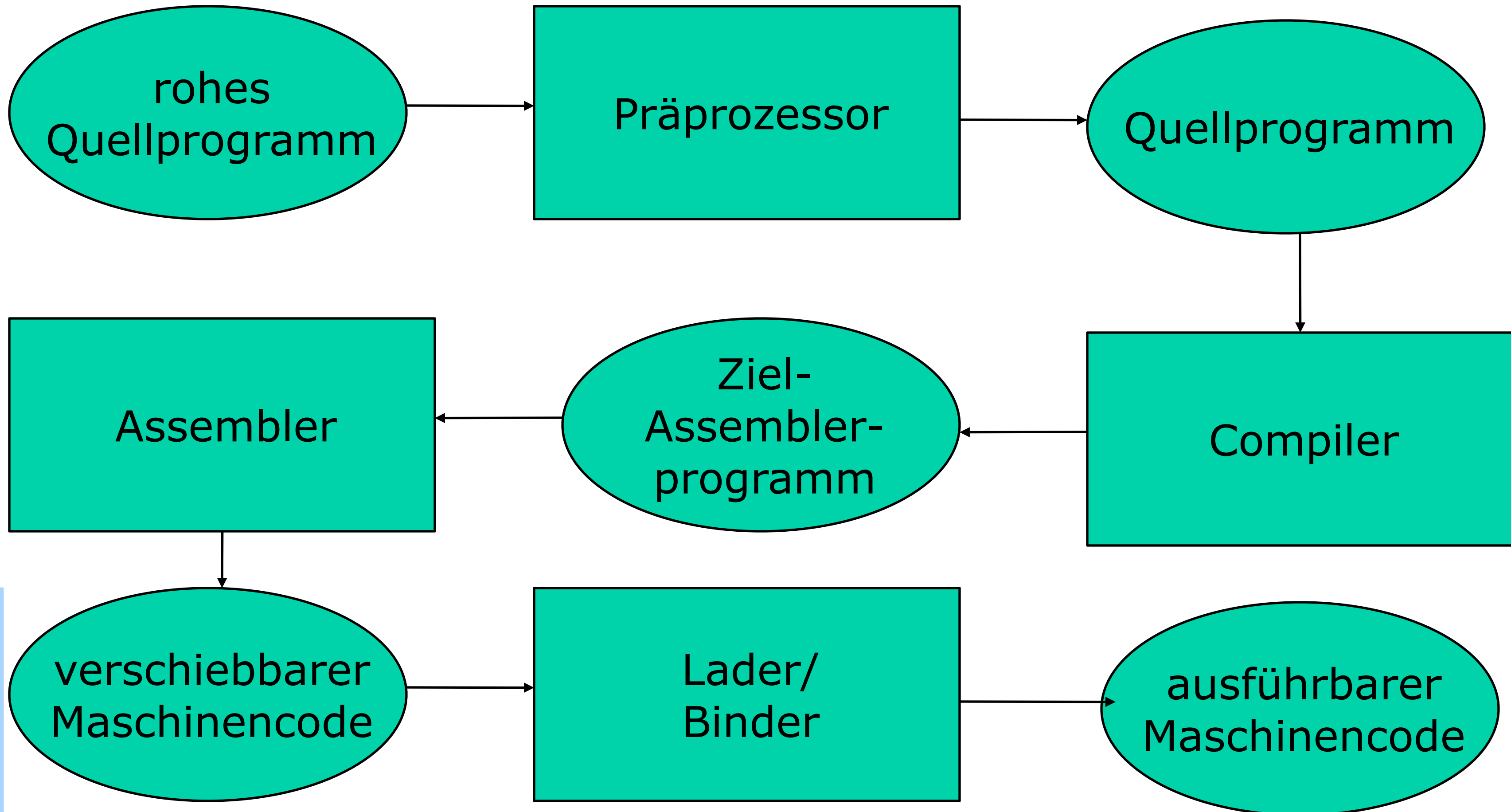
© 2006 Martin v. Löwis

Modell eines Compilers: Analyse/Synthese



© 2006 Martin v. Löwis

Modell eines Compilers: Compiler-Umgebung



© 2006 Martin v. Löwis

Analysephase

- 3 Teile
  - Lineare Analyse: Bytes/Zeichen werden von links nach rechts gelesen und in *tokens* gruppiert
  - Hierarchische Analyse: Tokenfolgen werden zu ineinander geschachtelten Gruppen zusammengefasst
  - Semantische Analyse: die inhaltliche Konsistenz der Programmbestandteile wird überprüft

© 2006 Martin v. Löwis

Lineare Analyse: Lexik

- lexikalische Analyse (*scanning*, *Scanner*, *Tokenizer*, *Lexer*)
  - oft formalisiert durch eine reguläre Grammatik
  - Scannergenerator erzeugt Scanner aus formaler Beschreibung
- üblicherweise kontextunabhängig
- Konfliktbewältigung (Mehrdeutigkeit) durch natürliche Sprache
  - Spezialbehandlung von Schlüsselwörtern (Konflikt mit Bezeichnern)
  - "*longest match*" ("1.3e9" vs. "1.3" "e9")
- Eliminierung von "irrelevanten" Tokens
  - Freiraum, Kommentare

© 2006 Martin v. Löwis

Lineare Analyse: Lexik (2)

- position := initial + rate * 60
  - Bezeichner: “`position`”
  - Zuweisungsoperator (:=)
  - Bezeichner: “`initial`”
  - Additionsoperator (+)
  - Bezeichner: “`rate`”
  - Multiplikationszeichen (*)
  - Ganzzahlliteral: “`60`”

© 2006 Martin v. Löwis

Hierarchische Analyse: Syntax

- Syntaxanalyse (*parsing*, *Parser*)
- induktive Definition von “Symbolen”
  - Hilfssymbole: Zusammenfassung von anderen Symbolen
  - Terminalsymbole: Tokens
- Zusammenfassung formalisiert in Grammatik
  - üblicherweise kontextfreie Grammatik
  - Parsergenerator erzeugt Parser aus formaler Beschreibung
- Ergebnis der Syntaxanalyse ist ein Parse-Baum

© 2006 Martin v. Löwis

Hierarchische Analyse: Syntax (2)

- Jeder *Bezeichner* ist ein *Ausdruck*
- Jedes Ganzzahlliteral ist ein Ausdruck
- Wenn $Ausdruck_1$ und $Ausdruck_2$ Ausdrücke sind, dann sind auch

  $Ausdruck_1 + Ausdruck_2$

  $Ausdruck_1 * Ausdruck_2$

  $(Ausdruck_1)$

  Ausdrücke

© 2006 Martin v. Löwis

Hierarchische Analyse: Syntax (3)

- Übliche syntaktische Kategorien
  - Ausdrücke (Vorrang entweder durch zahlreiche Hilfssymbole oder informal definiert)
  - Anweisungen
  - Definitionen (Funktionen, Datentypen, …)

© 2006 Martin v. Löwis

Semantische Analyse

- Bedeutung in Sprachdefinition: well-formedness
- Bedeutung in Compilerbau: Attributierung des Parse-Baums
  - Typisierung
  - Auflösen von Bezeichnern, Symboltabelle

© 2006 Martin v. Löwis

Phasen eines Compilers

- lexikalische Analyse
- Syntaxanalyse
- semantische Analyse
- Zwischencodeerzeugung
- Code-Optimierung
- Code-Erzeugung
- phasen-übergreifend:
  - Symboltabellenverwaltung
  - Fehlerbehandlung

© 2006 Martin v. Löwis

Symboltabellen

- Speicherung von Information über Bezeichner
  - Name des Bezeichners
  - Sichtbarkeit (*Scope*)
  - Art des Bezeichners (Variable: welcher Typ, Funktion: welche Argumente, welcher Rückgabetyp)
- Zwei Operationen: Eintragen und Nachschlagen (*lookup*)
  - lexikalische Analyse muss zwischen Template- und Nicht-Template-Namen unterscheiden (C++)
  - Code-Generierung muss zwischen Instanz- und statischen Variablen unterscheiden (Java)

© 2006 Martin v. Löwis

Fehlerbehandlung

- Fehlerarten abhängig von Compilerphase:
  - Syntaxfehler
  - Ill-formedness (Typfehler, undefinierte Bezeichner, …)
  - Überschreitung von Compilereinschränkungen (etwa: maximale Bezeichnerlänge, Verwendung nicht-unterstützter Konstrukte)
- Bezug zu Quelltext (Datei, Zeilen-, Spaltennummer)
- Diagnose möglichst in Begriffen der Programmiersprache (nicht in Begriffen des Compilerwerkzeugs)
- Optional: Warnungen (Analyse missverständlicher Sprachkonstrukte)
- Optional: Fehlerstabilisierung (Fortsetzen bei unkritischen Fehlern)

© 2006 Martin v. Löwis

Interne Darstellung

- Zwischencode: Code einer abstrakten Maschine
  - Stack-Maschine
  - Drei-Address-Code
- Abstrakte Syntax: Baum-Repräsentation einer abstrakten Sprache
  - Gemeinsame Repräsentation verschiedener Programmiersprachen
  - Abstraktion von lexikalischen und syntaktischen Details
    - Ersetzen von Interpunktion durch Baumstruktur
    - Ersetzen von Bezeichnern (lexikalischen Querverweisen) durch Pointer
- Meta-Modell: Konzepte der Programmiersprache

© 2006 Martin v. Löwis

Optimierung und Synthese

- Optimierung: Transformation einer internern Repräsentation in eine andere
  - Berechnung konstanter Ausdrücke zur Compile-Zeit
  - Eliminierung nicht erreichbaren Codes (dead code elimination)
  - …
- Synthese: Transformation der internen Repräsentation in Ausgabeformat
  - “manuelles” Traversieren der internen Struktur
  - muster-getriebene Synthese

© 2006 Martin v. Löwis

Literatur

- Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman: Compilerbau

© 2006 Martin v. Löwis