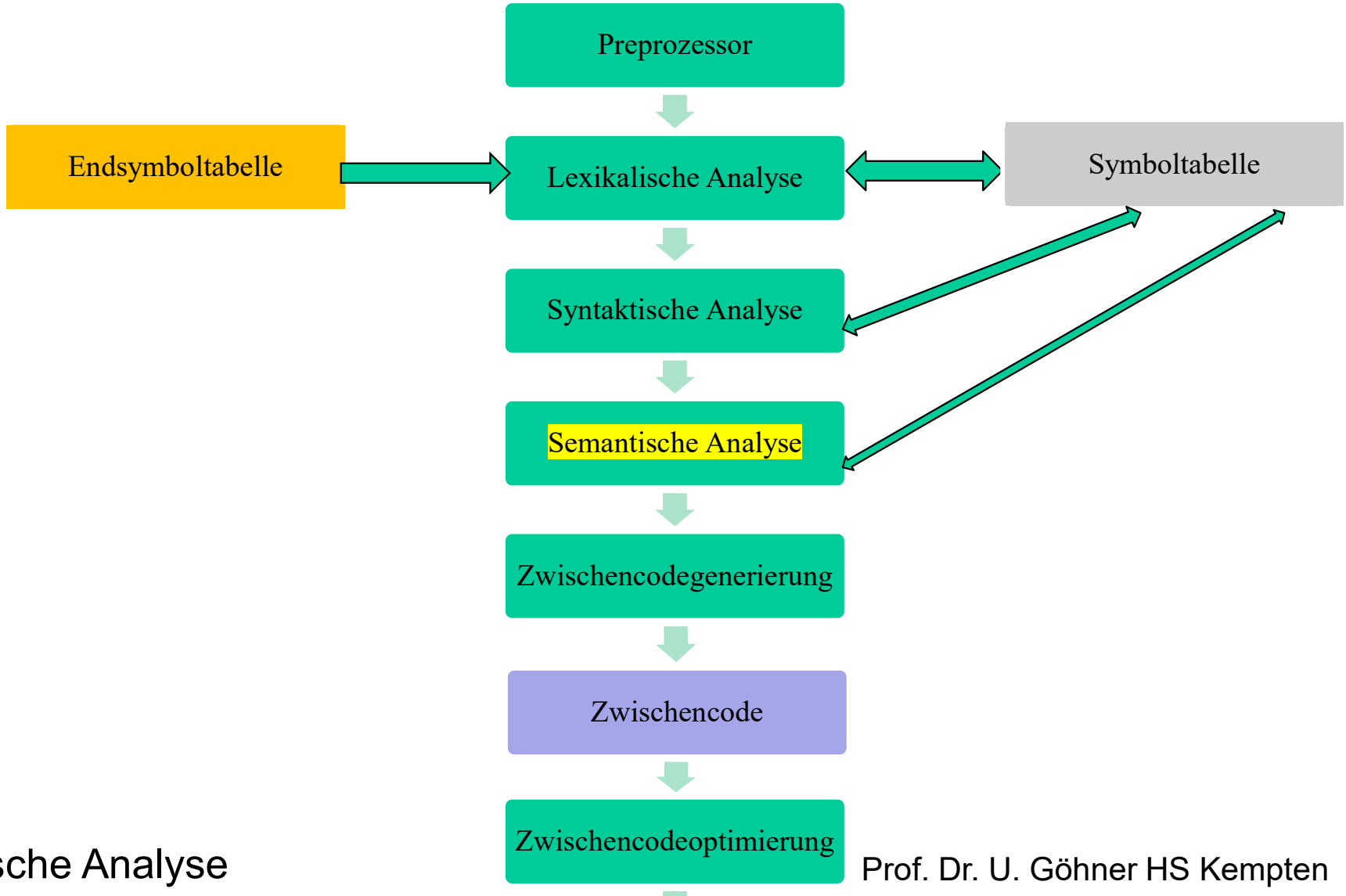
Semantische Analyse

Semantische Analyse



Semantische Analyse

Semantische Analyse

Übersetzungsprozess mit erforderlichen Berechnungen z.B.

- Überprüfung von Typkompatibilitäten
- Erzeugung Zwischen-Code

Semantische Analyse

Syntaxorientierte Übersetzung:

- Die Algorithmen für die nötigen Berechnungen werden eindeutig den Syntaxregeln der Grammatik zugeordnet

Vorteile: der syntaxorientierten Übersetzung

- flexibler Mechanismus
- automatische Generierung leicht möglich

Semantische Analyse

Definition Attributierte Grammatik:

1. Attribut ist eine Information, die einem Knoten im Syntaxbaum zugeordnet ist.
2. Die für die Übersetzung und weitere Berechnung benötigten Werte können als Attribute der Baumknoten definiert werden.

Beispiele:

- Berechnung der Variablentypen in einer Ausdrucksgrammatik:
  - Jedem Knoten im Baum wird das Attribut “Typ” zugeordnet

Definition attributierter oder dekorierter Baum:

Ein Syntaxbaum mit berechneten Attributwerten heißt attributierter oder dekorierter Baum.

Semantische Analyse

Vererbte und synthetische Attribute

- Attributwerte, die Vaterknoten an die Söhne übergeben werden, heißen vererbt (top-down-Berechnung)
- Attributwerte, die vom Sohn zum Vater übergeben werden, heißen synthetisch (bottom-up-Berechnung).

Semantische Analyse

Eine attributierte Grammatik ist eine Grammatik G = (T, N, S, P) mit folgenden Erweiterungen:

- jedem Symbol $X \in T \cup N$ ist eine endliche Menge von Attributen $\{X.A_1, X.A_2, ..., X.A_k\}$ zugeordnet.
- Zu jeder Ableitungsregel X -> w wird angegeben, wie die synthetischen Attribute von X zu berechnen sind.
- Zu jeder Ableitungsregel X ->w wird angegeben, wie die vererbten Attribute zu berechnen sind.
- Notation:
  - Schreibweise $X.\uparrow A_i$ für synthetisches Attribut
  - $X.\downarrow A_i$ für vererbtes Attribut

Semantische Analyse

Beispiel für eine attributierte Grammatik:

- Roboter mit Befehlen N, O, S, W für schrittweise Bewegung in eine der vier Himmelsrichtungen
- Ausgangspunkt Koordinaten x = 0 und y = 0
- zu jeder Befehlsfolge soll die erreichte Endposition berechnet werden.

z.B. NNNOO -> x=2, y=3 oder SSW -> x=-1, y=-2

Semantische Analyse

Semantische Aktionen

- Beim Ausführen der rechten Regelseite werden zwischen den Grammatik-Symbolen semantische Aktionen eingefügt.
- Semantische Aktionen können beliebige Berechnungen spezifizieren.
- Die Reihenfolge der Berechnungen ist durch die Reihenfolge der semantischen Aktionen in der Regel bestimmt.
- Semantische Aktionen können bei der automatische Generierung des Parsers leicht generiert werden

Semantische Analyse

Ausführung der Aktionen beim Top-Down-Parser

- Top-Down-Parser wählt eine Ableitungsregel aus, z.B.

  $S \rightarrow a_1 .... a_k$

- bearbeite Bestandteile der rechten Seite $a_1 .... a_k$ von links nach rechts.
- neben Terminal- und Nonterminalzeichen werden auch Aktionen als Bestandteile der rechten Regelseite zugelassen
- Bearbeiten der Aktion heißt das Ausführen der Aktion.

Semantische Analyse

Ausführung der Aktionen beim Bottom-Up-Parser:

1. Regelende-Aktionen:

- Ist letzter Bestandteil der rechten Regelseite
- Wird bei der Reduktion der Regel ausgeführt

2. Regelmitte-Aktionen:

- Überall sonst, aber nicht am Ende
- Unklar, wann Aktion ausgeführt werden soll

Semantische Analyse

Behandlung von Regelmitte-Aktionen

- Aktion A sei folgende Regelmitte-Aktion:

X -> u {A} v

Aktion A wird nach dem Lesen von u und vor der Verarbeitung von v ausgeführt

- Der Bottom-Up-Parser kann die Reduktion nur gesamt durchführen, nicht teilweise
- Regelmitte-Aktion muss deshalb in eine Regelende-Aktion umgewandelt werden:
  - neues Nonterminalsymbol N_A wird eingeführt. N_A repräsentiert die Aktion: $X \rightarrow uN_Av$
  - für N_A wird eine neue Regel mit Regelende-Aktion eingeführt:

    $N_A \rightarrow \varepsilon\{A\}$

Semantische Analyse

yacc-Notation für Attributberechnung:

- yacc unterstützt synthetische Attribute.
- jedem Grammatik-Symbol ein Attribut zugewiesen werden
- Die Attributwertberechnung für Nonterminalsymbole wird durch semantische Aktionen spezifiziert.
- Eine Aktion ist eine Anweisung, die in der rechten Seite einer Ableitungsregel steht.
- In der Aktion kann auf die Attribute zugegriffen warden

  $$ = Attribut der linken Regelseite

  $1, $2 usw. Attribute des 1., 2., . . . Symbols auf der rechten Regelseite.

Semantische Analyse

Generieren von Steuerinformationen für yacc:

Token-Definition, z.B.:

%token ZAHL ADD SUB

Es werden #define–Anweisungen erzeugt und in ein header-file geschrieben

#define-Anweisungen können in der lexikalischen Analyse verwendet werden. (Einlesen des header-files im flex-file)

- Attributwerte der Tokens können beim Scanner berechnet werden.

  Werte werden über die externe Variable yylval übergeben

  z.B.: yylval=atoi(yytext) im flex-file bestimmen und an Parser bison übergeben

Semantische Analyse

Steuerinformationen für yacc:

- Für Variablen mit anderen Typen als „int“ ist yylval vom Typ „union“ in der Steuerinformation als %union zu definieren
- Zur Typisierung der Tokens wird dem Parser der Token-Typ in der %token–Anweisung mitgeteilt.
- Der union Typ wird in spitzen Klammern genannt
- In den Aktionen des Scanners erfolgt die Weitergabe der erkannten Werte in den Komponenten der externen Variable yylval
- Andere Typen als int für Nonterminals werden im Deklarationsteil mit

  %type <Typ> Nonterminal-Symbol

  festgelegt

  <Typ> muss durch %union definiert sein)

Semantische Analyse

Grammatik und Aktionen in yacc:

- Produktionen sind in EBNF anzugeben
- am Ende einer Produktion werden die semantische Aktionen (Attributberechnungsregeln) angegeben
- Aktionen werden in geschweifte Klammern gesetzt.
- Aktionen werden ausgeführt, wenn nach der entsprechenden Produktion reduziert wird.
- Aktion besteht aus beliebigen C-Anweisungen oder Berechnungsregelen (Zuweisungen/Berechnungen der Attributwerte)
- Zugriff auf die Attributwerte der Symbole der rechten Seite: $1, $2,...
- Zugriff auf die Attributwerte der Symbole der linken Seite: $$

Semantische Analyse

In Yacc definierte Funktionen:

- Funktion yyparse() startet den Parse-Vorgang
- Fehlerbehandlung yyerror():
  - kann durch eine eigene Funktion ersetzt werden
  - Ergänzen der Fehlermeldung durch weitere Informationen wie z.B. Position des Scanners
- Literatur: Herold: Linux-Unix-Profitools, Levine et al.: lex & yacc

Semantische Analyse

Symboltabellen-Einträge

- Anhand der Bezeichnerklasse wird bestimmt, ob der Bezeichner Variablen, Typen, Prozeduren oder andere Objekte repräsentieren.
- Bezeichnereintrag muss den Namen und die Klasse enthalten
- je nach Klasse werden weitere Attribute für die Übersetzung benötigt

Semantische Analyse

Symboltabelle

- Symboltabelle enthält zu jedem Bezeichner einen Eintrag mit allen relevanten Attributen.
- Der Eintrag wird bei der Analyse der Definitionsstelle des Bezeichners erzeugt
- Bei jeder Verwendungsstelle eines Bezeichners liest Compiler den Symboltabelleneintrag, prüft die korrekte Verwendung, ergänzt die Symboltabelle und erzeugt ggf. entsprechenden Code

Semantische Analyse

Symboltabellen-Operationen:

- neue Tabelle erzeugen: S = new Sym_tab;
- Tabelle löschen: delete S;
- neuen Eintrag erzeugen: S.add(ID, Klasse)
- Eintrag für Bezeichner ID in einer Tabelle S suchen:
  Eintrag = S.lookup(ID)
- Attribut setzen: S.set_attribut_x (value)
- Attribut lesen: value = S.get_attribut_x ()
- Nach der Compilierung wird die Symboltabelle gelöscht.
  Ausnahme: Für Debug-Zwecke wird Symboltabelle auch bei Ausführung benötigt

Semantische Analyse

Symboltabellen und Gültigkeitsbereiche

- Für jeden Gültigkeitsbereich wird eine neue Symboltabelle angelegt
- Beispiel C:
  - globale Symboltabelle mit den Einträgen für globale Variablen, Typbezeichner und Funktionsbezeichner
  - In Funktionsdefinition wird eine neue Symboltabelle mit den lokalen Bezeichnern angelegt.

Semantische Analyse

Implementierung der Symboltabelle

- Symboltabelle lässt sich z.B. als verkettete Liste von Symboltabellen-Einträgen implementieren.
- Da die Suche eines Bezeichnereintrags sehr häufig benötigt wird, sollte ein effizientes Suchverfahrens zur Verfügung stehen (z.B. Hashverfahren, binärer Suchbaum,...)

Analyse-Tools

- Flex+bison
- VCC
- OOLEX/OOPS
  - Object-Oriented Lexer/Parser
  - Java
- ANTLR3 (C, C#, Java,…)
  - GUI-Unterstützung
  - Debugging

Code-Erzeugung

Maschinencode-Generierung und Optimierung

Abstrakte Maschinen und Zwischencode-Generierung

- Ein-Phasen-Compiler erzeugt den Code direkt während der einphasigen Analyse
- 2-Phasen-Compiler bestehen aus Frontend und Backend:
  - Frontend erzeugt den Syntaxbaum
  - Backend erzeugt Code auf Basis des Syntaxbaums
- bei der Code-Erzeugung wird in vielen Fällen statt Maschinencode sondern zunächst Zwischencode erzeugt
- Zwischencode basiert in der Regel auf den Datenstrukturen und dem Befehlssatz einer abstrakten virtuellen Maschine

Code-Erzeugung

Abstrakte Maschinen und Zwischencode-Generierung

Vorteile Zwischencode:

- Optimierungen auf hoher Abstraktionsebene möglich
- bei Portierung auf andere Hardware-Maschine ist nur das maschinenabhängige “backend” anzupassen
- Zwischencode kann auch interpretiert werden
- Bekannte Zwischencodes sind:
  - 3-Adress-Code (prozedurale Sprachen)
  - Stackmaschinencode
  - SSA-Code (Static-Single-Assignment)

Code-Erzeugung

Code-Erzeugung für Stack-Maschinen

- Stack-Maschine benutzt für die Speicherung von Zwischenergebnissen einen Stack
- Stack kann durch Register der CPU nachgebildet werden

Code-Erzeugung für Registermaschinen

- Prozessor kann 2- oder 3-Adresscode verarbeiten
- Zwischencode benutzt symbolische Adressen

Code-Optimierung

Ziele:

- schnellere Maschinenprogramme
- Wenig Speicherverbrauch
- kleinere Maschinenprogramme

Optimizerungsziele können sich widersprechen

Übersetzung dauert um so länger, je stärker der Code optimiert wird (meist über Compiler Options regelbar)

Teilweise bei sensitiven Berechnungen andere Resultate möglich(!)

Code-Optimierung

Maschinenabhängige Code-Optimierung hängt von der Architektur der Ziel-Hardware ab:

- Anzahl der Register
- Zugriffszeit auf Hauptspeicher
- Möglichkeit der Parallelverarbeitung
- …..

Code-Optimierung

Zwischencode-Optimierung:

folgende Methoden werden häufig eingesetzt:

- Berechnung der Konstanten („constant folding“)
- Variablenersatz in Zuweisungen („constant propagation“)
- Entfernen von nicht durchlaufenem Code
- Expandieren von Schleifen
- Zusammenfassung sukzessiver Zuweisungen
- Unterdrückung von Mehrfachberechnungen
- Entfernen unnötiger Zuweisungen
- Verschiebung von Schleifeninvarianten
- Zusammenführen von Konstanten gleichen Inhalts
- Optimierung arithmetischer Ausdrücke