© http://www.ausmalbilder.info

02-07-2013 Wolfgang Dichler

BoB: Best of Both in Compiler Construction

Combination of Bottom-up Syntax Analysis and Top-down Semantic Analysis

students@fh-ooe

Inhalt

1 Einleitung

2 Architektur

3 Lexikalische Analyse

4 Syntaktische Analyse

5 Semantische Analyse

6 Abschluss

Hintergrund/Motivation

Zwei vorherrschende Strategien für die SYNTAX-Analyse

- Top-down
- Bottom-up

Existierende Compiler-Generatoren gehen Kompromisse

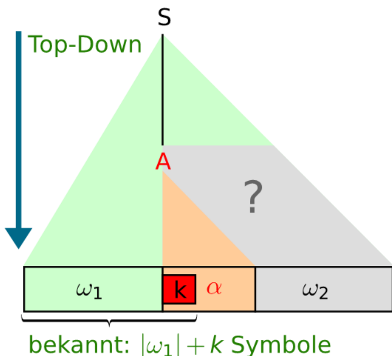
- bei der Mächtigkeit der unterstützten Grammatik &
- bei den unterstützten Attributen in den semantischen Aktionen ein.

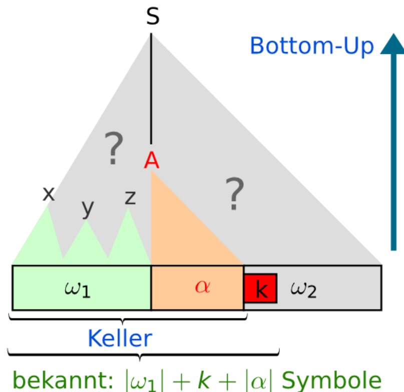
Problembeschreibung

LL(k)-Analyse vs. LR(k)-Analyse

$$S \Rightarrow \omega_1\, A\, \omega_2 \overset{?}{\Rightarrow} \omega_1\, \alpha\, \omega_2 \Rightarrow \sigma$$

$$\sigma \vdash \omega_1\, \alpha\, \omega_2 \overset{?}{\vdash} \omega_1\, A\, \omega_2 \vdash S$$





©Formale Sprachen u. Automatentheorie, H. Dobler

Stand der Technik

Compiler Generatoren:

- Top-down-Analyse – Coco-2
- Bottom-up-Analyse – Flex/GNU Bison
- Bottom-up- / Top-down-Analyse – Smart

Compiler Beschreibungssprachen:

- Cocol-2
  - EBNF für Beschreibung des Scanners und Parsers
- Flex/Bison
  - Reguläre Ausdrücke zur Beschreibung des Scanners
  - BNF zur Beschreibung der Regeln des Parsers

Zielsetzung

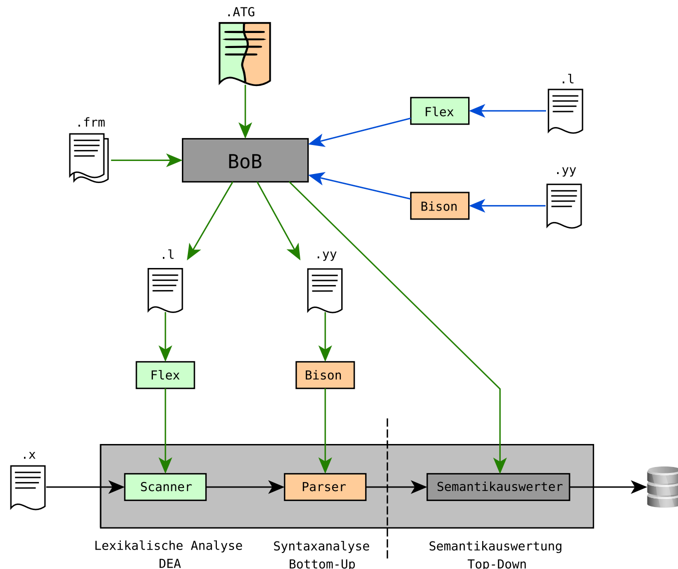
Implementieren eines Präprozessors, der die Vorteile eines Bottom-up-Syntaxanalysators mit den Vorteilen eines Top-down-Semantikauswerters vereint:

- Grammatik muss nur LALR(1) genügen
- Unterstützung von Eingangs-, Ausgangs- und Übergangsattributen

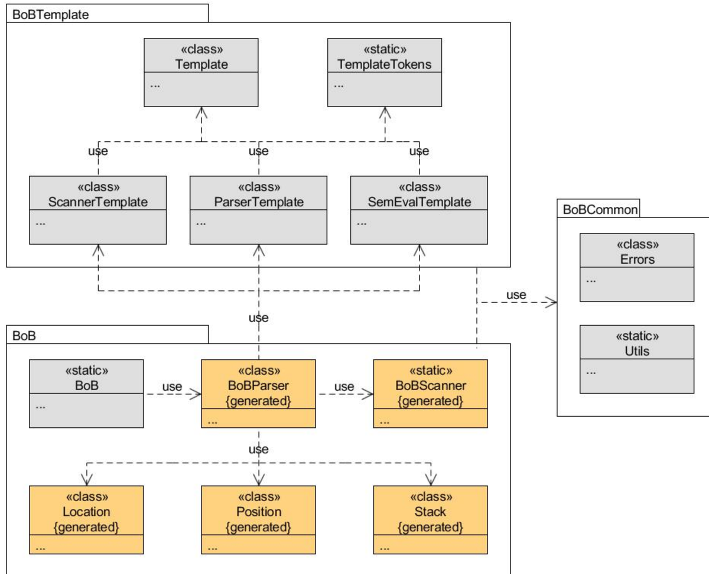
Verwendung von bereits etablierten Technologien:

- Compilerbeschreibungssprache: basierend auf Cocol-2
- Scanner und Parser: Flex und GNU Bison

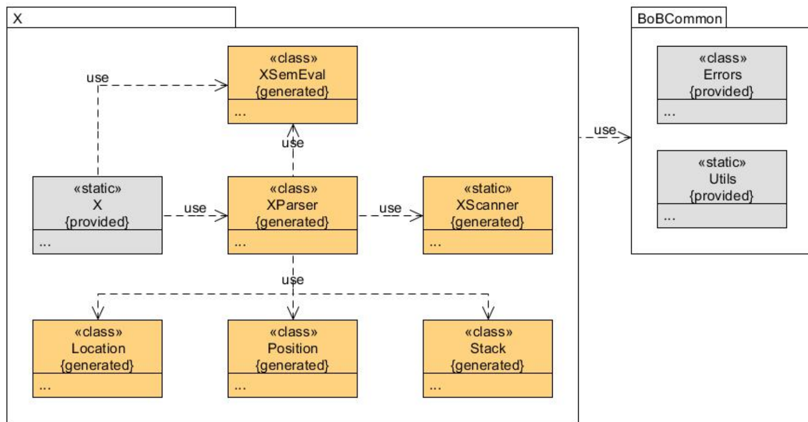
Lösungsansatz



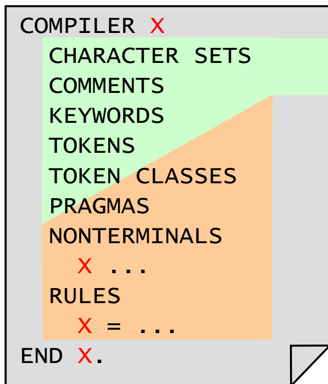
BoB

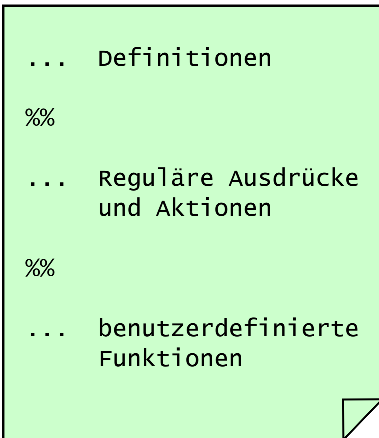


Generierte Compiler



Überblick





© In Anlehnung an Compiler-Generatoren und Coco-2 für FCW1, H. Dobler

Zeichenmengen

Regeln für Zeichenmengen

Cocol4BoB

```
1 SetDecl = ident<|str|> '='
2           ChrSet { ( '+' | '-' ) ChrSet } [ 'IGNORE' ] '.' .
3 ChrSet  = ident<|str|> | 'ANY' | 'EOL'
4           | Special [ '..' Special ] .
5 Special = str<|str|>
6           | 'CHR' '(' num<|num|> ')'.
```

Zuordnungstabelle

Cocol4BoB	Flex-Input (Regex)
Special .. Special	[Special-Special]
ANY	[^\n]
EOL	[\n]
CHR(0 . . . 127)	\x00, . . . , \x7f
$+$	{+}
$-$	{−}

Zeichenmengen

Beispieldefinitionen von Zeichenmengen — Cocol4BoB

```
1 CHARACTER SETS
2     letter      = 'A'..'Z' + 'a'..'z'.
3     digit       = '0'..'9'.
4     control     = CHR(0) .. CHR(31) + CHR(127).
5     noQuote     = ANY - "'" - control.
6     ignore      = EOL + CHR(9) IGNORE.
```

Ergebnis der Transformation — Flex-Input

```
1     letter         [A-Z]{+}[a-z]
2     digit          [0-9]
3     control        [\x00-\x1f]{+}[\x7f]
4     noQuote        [^\n]{-}[']{-}[\x00-\x1f]{-}[\x7f]
5     ignore         [\n]{+}[\t]
```

Zeichenmengen

Beispieldefinitionen von Zeichenmengen

Cocol4BoB

```
1 CHARACTER SETS
2     letter      = 'A'..'Z' + 'a'..'z'.
3     digit       = '0'..'9'.
4     control     = CHR(0) .. CHR(31) + CHR(127).
5     noQuote     = ANY - "'" - control.
6     ignore      = EOL + CHR(9) IGNORE.
```

Ergebnis der Transformation

Flex-Input

```
1     letter      [A-Z]{+}[a-z]
2     digit       [0-9]
3     control     [\x00-\x1f]{+}[\x7f]
4     noQuote     [^\n]{-}[']{-}[\x00-\x1f]{-}[\x7f]
5     ignore      [\n]{+}[\t]
```

Kommentare

Regeln für Kommentare

Cocol4BoB

```
1 CommentDecl   = 'FROM' Delimiter 'TO' Delimiter
2                 [ 'NESTED' ] [ 'HANDLER' ident<|str|> ] '.' .
3 Delimiter     = DelimiterPart [ DelimiterPart ] .
4 DelimiterPart =   str<|str|>
5                 | ident<|str|>
6                 | 'CHR' '(' num<|num|> ')'
7                 | 'EOL' .
```

Beispieldefinitionen von Kommentaren

Cocol4BoB

```
1 COMMENTS
2   FROM '//' TO EOL         HANDLER CmntHandler. // C++
3   FROM '/*' TO '*/'        HANDLER CmntHandler. /* C          */
4   FROM '(*' TO '*)' NESTED HANDLER CmntHandler. (* Modula-2 *)
```

Kommentare

Einzeilige Regex-Ausdrücke???

Flex-Input

```
1   "/*"([^\*]|\*[^/])*"*/"
2   /\/\*.*?\*\//s
```

Kommentare

Einzeilige Regex-Ausdrücke??? — Flex-Input

```
1   "/*"([^\*]|\*[^/])*"*/"
2   /\/\*.*?\*\//s
```

Ergebnis – Zeilen-Kommentare — Flex-Input

```
1 %x LINE_CMNT
2
3 "//" { BEGIN(LINE_CMNT); ResetCommentHandler(); }
4
5 <LINE_CMNT>{ /* Line Comment */
6   [^\n]+  { DefaultCmntHandler(CmntHandler); }
7   [\n]    { BEGIN(INITIAL); }
8 }
```

Kommentare

Einzeilige Regex-Ausdrücke???

Flex-Input

```
1  "/*"([^\*]|\*[^/])*"*/"
2  /\/\*.*?\*\//s
```

Ergebnis – Zeilen-Kommentare

Flex-Input

```
1 %x LINE_CMNT
2
3 "//" { BEGIN(LINE_CMNT); ResetCommentHandler(); }
4
5 <LINE_CMNT>{ /* Line Comment */
6   [^\n]+  { DefaultCmntHandler(CmntHandler); }
7   [\n]    { BEGIN(INITIAL); }
8 }
```

Kommentare

Ergebnis – Block-Kommentare

Flex-Input

```
1  %x BLOCK_CMNT
2
3  "/*" { BEGIN(BLOCK_CMNT); ResetCommentHandler(); }
4
5  <BLOCK_CMNT>{ /* Block Comment */
6    <<EOF>> { /* Error EOF within comment */ }
7    [^*\n]+ { DefaultCmntHandler(CmntHandler); }
8    "*"     { DefaultCmntHandler(CmntHandler); }
9    [\n]+   { DefaultCmntHandler(CmntHandler); }
10   "*/"    { BEGIN(INITIAL); }
11 }
```

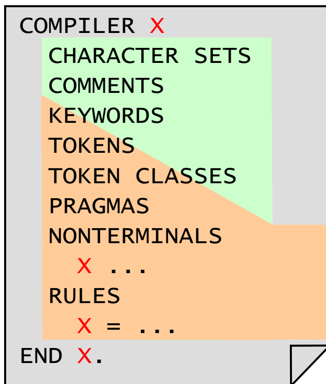
Kommentare

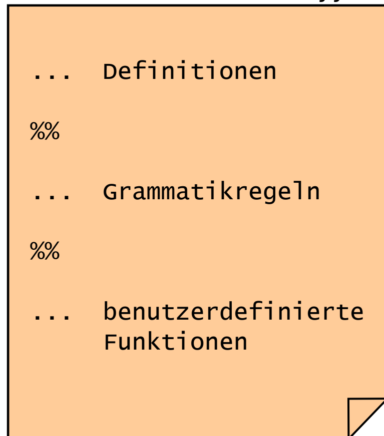
Ergebnis – Verschachtelte Block-Kommentare

Flex-Input

```
1  %x NESTING_COMMENT
2
3  "(*"    { BEGIN(NESTING_COMMENT); ++depth; ResetCmntHandler(); }
4
5  <NESTING_COMMENT>{ /* Nested Block Comment */
6    <<EOF>>    { /* Error EOF within comment */ }
7    "(*"       { ++depth; DefaultCmntHandler(CmntHandler); }
8    [^(*\n]+   { DefaultCmntHandler(CmntHandler); }
9    "("        { DefaultCmntHandler(CmntHandler); }
10   "*"        { DefaultCmntHandler(CmntHandler); }
11   [\n]+      { DefaultCmntHandler(CmntHandler); }
12   "*)"       { --depth;
13                if(depth == 0) { BEGIN(INITIAL); }
14                else { DefaultCmntHandler(CmntHandler); }
15              }
16 }
```

Überblick





© In Anlehnung an Compiler-Generatoren und Coco-2 für FCW1, H. Dobler

Grammatikregeln

Ersetzen der EBNF-Konstrukte (), [] und {} durch künstliche Nonterminalsymbole.

Transformation von EBNF zu BNF

EBNF	BNF	
	Variante 1	Variante 2
A = a (b|c).	$A \rightarrow$ a X $X \rightarrow$ b | c	
A = a [b].	$A \rightarrow$ a X $X \rightarrow$ b | ε	$A \rightarrow$ a | a X $X \rightarrow$ b
A = a {b}.	$A \rightarrow$ a X $X \rightarrow$ X b | ε	$A \rightarrow$ a | a X $X \rightarrow$ b | X b

Grammatikregeln

Beispiel für LR(1)-Konflikt bei Variante 1

EBNF	BNF	
	Variante 1	Variante 2
A = a b | {a} c.	A $\rightarrow$ a b | X c X $\rightarrow$ X a | ε	A $\rightarrow$ a b | c | X c X $\rightarrow$ a | X a

Lesen des Bandsymbols *a* führt bei Variante 1 zu einem Shift/Reduce-Konflikt:

- Shift (nächstes Bandsymbol lesen): A $\rightarrow$ a b oder
- Reduce (ε zu X reduzieren): X $\rightarrow$ X a

Grammatikregeln

Beispieldefinition einer Grammatikregel

Cocol4BoB

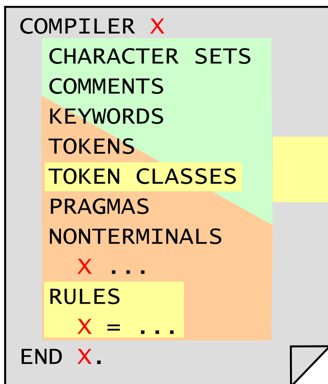
```
1 SetDecl =    ident<|str|> '='
2              ChrSet { ( '+' | '-' ) ChrSet } [ 'IGNORE' ] '.' .
```

Ergebnis der Transformation nach BNF

Bison-Input

```
1  SetDecl_1:
2     '+' | '-' ;
3
4  SetDecl_2:
5      /* epsilon */ | SetDecl_2 SetDecl_1 ChrSet ;
6
7  SetDecl_3:
8      /* epsilon */ | IGNORE ;
9
10 SetDecl:
11    ident '=' ChrSet SetDecl_2 SetDecl_3 '.' ;
```

Übersicht



Für Semantic Analyzer
in Datei XSem.cpp:

... Lexikalische Daten (Datenstruktur)

... Reduktionsfolge (Datenstruktur)

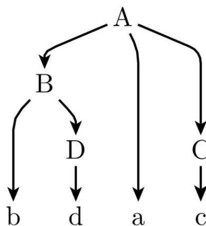
... semantische Methoden

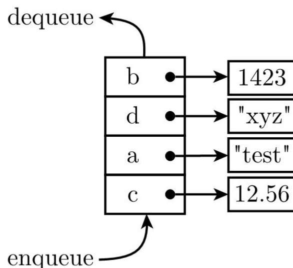
© In Anlehnung an Compiler-Generatoren und Coco-2 für FCW1, H. Dobler

Lexikalische Daten

- Repräsentieren die Attributwerte der Terminalsymbole.
- Erzeugung während der Bottom-up-Syntaxanalyse.
- Daten werden in Infix-Form benötigt. Können sofort beim Lesen durch den Scanner in eine FIFO-Queue gespeichert werden.

$$
\begin{aligned}
A &= B\ a_{\uparrow val}\ C. \\
B &= b_{\uparrow val}\ D. \\
B &= x_{\uparrow val}\ C. \\
C &= c_{\uparrow val}. \\
D &= d_{\uparrow val}.
\end{aligned}
$$



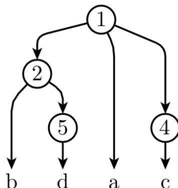


© In Anlehnung an Compilererzeugende Systeme für Mikrocomputer, H. Mössenböck

Reduktionsfolge

- Repräsentiert die bei der Analyse besuchten Regeln.
- Erzeugung während der Bottom-up-Syntaxanalyse.
- Minimaler Aufwand an Speicherplatz und Rechenzeit.
- Transformation von Postfix- in Präfix-Form.

(1) A = B a C.
(2) B = b D.
(3) B = x C.
(4) C = c.
(5) D = d.



Reduction Sequence	Post-Order	5	2	4	1
	Pre-Order	1	2	5	4

NT-Sons	0	2	1	1	0	0
Reduction Number	0	1	2	3	4	5

© In Anlehnung an Compilererzeugende Systeme für Mikrocomputer, H. Mössenböck

Semantische Methoden

Grammatikregel — Cocol4BoB

```
1  A<|[1]|> =
2    LOCAL<|[2]|>
3    PRE  <|[3]|>
4    POST <|[4]|>
5    a<|[6]|> SEM<|[5]|>
6    B<|[7]|> SEM<|[5]|>    (1)
7  | ( b<|[6]|> (5) )       (2)
8  | [ C<|[7]|> (6) ]       (3)
9  | { c<|[6]|> (7)
10   | d<|[6]|> (8) }       (4)
11 .
```

- (1)–(8) Reduktionsnummern
- [1] Formalparameter der semantischen Methode
- [2] Lokale Deklarationen
- [3] Semantische Aktionen – PRE
- [4] Semantische Aktionen – POST
- [5] Semantische Aktionen – INTERLEAVED
- [6] Aktualparameterliste – Terminal Symbole
- [7] Aktualparameterliste – Nonterminal Symbole

Semantische Methoden

Sem. Methode – Teil 1 (C++)

```cpp
1  void A([1]) {
2   [2] [3]
3   bool done = false;
4   switch(nextRedNr()) {
5    case (1):
6      GETaAttr([6]); /*SEM*/ [5]
7      B([7]);        /*SEM*/ [5]
8      break;
9    case (2): // grouping
10     switch(nextRedNr()) {
11       case (5): GETbAttr([6]);
12         break;
13     } // switch
14     break;
15   case (3): // option
16     switch(nextRedNr()) {
17       case (0): break;
```

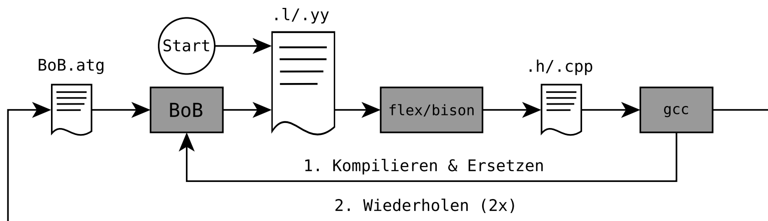
Sem. Methode – Teil 2 (C++)

```cpp
18         case (6): C([7]); break;
19     } // switch
20     break;
21   case (4): // repetition
22     while(!done) {
23       switch(nextRedNr()) {
24         case (0): done=true; break;
25         case (7): GETcAttr([6]);
26           break;
27         case (8): GETdAttr([6]);
28           break;
29       } // switch
30     } // while
31     break;
32  } // switch
33  [4]
34 } // A
```

Testdaten / Bootstrapping

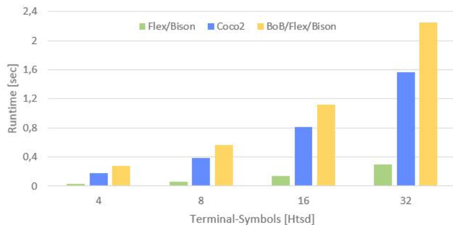
ATGs für Verifikation und Validierung

ATG	Größe	Compiler Generator	Zweck
BoB	Groß	*BoB*	Verifikation
Calc	Klein	*BoB*, *Coco-2*, *flex/bison*	Validierung
MiniC	Mittel	*BoB*, *Coco-2*, *flex/bison*	Validierung
MiniCpp	Groß	*BoB*, *Coco-2*, *flex/bison*	Validierung



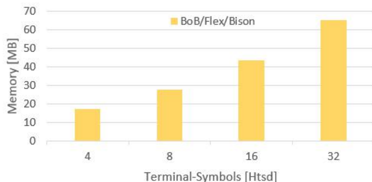
Performance

Laufzeit:



PROGRAM: MiniCpp
COMPILER: g++ (GCC) 4.7.2
TEST SYSTEM:
Intel® Core™2 Duo E6750
CLOCK SPEED: 2 x 2.66GHz
RAM: 4GB
OS: Windows® 8 64-Bit

Speicherbelegung (Flex/Bison und Coco-2 konstant 350KB):



PROGRAM: MiniCpp
COMPILER: g++ (GCC) 4.7.2
TEST SYSTEM:
Intel® Core™2 Duo E6750
CLOCK SPEED: 2 x 2.66GHz
RAM: 4GB
OS: Windows® 8 64-Bit

Zusammenfassung

BoB zeichnet sich durch folgende Punkte aus:

- Verwendung von Cocol4BoB als Eingabesprache
  - Eine einzige Eingabedatei
  - EBNF für die Beschreibung von Scanner und Parser
- BoB unterstützt mächtige LALR(1)-Grammatiken
- BoB unterstützt in den semantischen Aktionen Eingangs-, Ausgangs- und Übergangsattribute

BoB

vereint die Vorteile der Bottom-up-Syntaxanalyse und der Top-down-Semantikauswertung unter Inkaufnahme einer schlechteren Performance der erzeugten Compiler.

students@fh-ooe

Vielen Dank für Ihre Aufmerksamkeit!

Fragen?

Kontakt:
Wolfgang Dichler
☎ +43 (680) 40 316 40
✉ w.dichler@gmail.com