Theoretical Computer Science

Word Problem & Parsing

Technische Hochschule Rosenheim
Sommer 2022
Prof. Dr. Jochen Schmidt

Overview

- Solving the word problem
- CYK-Parser
- Application: Compilers

Analysis of Words & Parsing

Word Problem and Parsing Problem

Technische Hochschule Rosenheim

- Word problem: Decide whether a word x is well-formed (i.e., part of given language)

- Parsing problem (*Zerteilungsproblem*)
  - complete analysis of the word x
  - derivation of x is traced back to the start symbol S or vice versa
  - all steps from $S \Rightarrow^* x$ must be determined

- Questions:
  - How do we solve these problems for the various Chomsky language classes?
  - How difficult/time-consuming is the decision?

Word Problem for Regular Languages (Type 3)

- Create a deterministic finite automaton for the language
- If the processing stops in an end state, the analyzed word is part of the language
- Time complexity: linear with respect to the word length

Word Problem for Context-free Languages (Type 2)

- Create a pushdown automaton? We may require a **non**deterministic PDA for type 2 …
- Better: Convert grammar to Chomsky normal form (*Chomsky-Normalform*) and use the CYK-Algorithm (Cocke, Younger, Kasami)
- Time complexity: $O(n^3)$ with respect to the word length
  - too slow for practical purposes (e.g., syntax analysis in a compiler)
  - in programming languages: restriction to deterministic context-free languages and LR(k) grammars
    - these have linear time complexity
    - and we can use a deterministic PDA for solving the word problem

Chomsky Normal Form – Definition

- For any context-free grammar G with $\varepsilon \notin L(G)$ we can construct a grammar G' with L(G) = L(G') that is in Chomsky normal form

- A grammar is Chomsky normal form (CNF) if all production rules have the form (A, B, C syntactic variables, a terminal symbol)
  - $A \longrightarrow BC$
  - $A \longrightarrow a$

- If $\varepsilon \in L(G)$: Add the rule $S \longrightarrow \varepsilon$, S must not appear on the right side of any production
  - this is already required by our definition of type 2 grammars

Chomsky Normal Form – Construction

Technische Hochschule Rosenheim

Example: $L = \{a^m b^n c^n \mid m, n \geq 1\}$

$S \longrightarrow AB,\ A \longrightarrow Aa \mid a,\ B \longrightarrow bBc \mid bc$

- For each terminal symbol a: Create a new variable V_a
  - add productions $V_a \longrightarrow a$ to the grammar
  - replace terminal symbols a on all right-hand sides by V_a

$V_a \longrightarrow a,\ V_b \longrightarrow b,\ V_c \longrightarrow c$

$S \longrightarrow AB,\ A \longrightarrow AV_a \mid V_a,\ B \longrightarrow V_bBV_c \mid V_bV_c$

- Replace rules with more than 2 variables on the right
  - the rule $A \longrightarrow B_1B_2 \ldots B_k,\ k \geq 3$ becomes:
  - $A \longrightarrow B_1V_2$
    $V_2 \longrightarrow B_2V_3$
    …
    $V_{k-1} \longrightarrow B_{k-1}B_k$

$V_a \longrightarrow a,\ V_b \longrightarrow b,\ V_c \longrightarrow c$

$S \longrightarrow AB,\ A \longrightarrow AV_a \mid V_a,\ B \longrightarrow V_bV_1 \mid V_bV_c,$

$V_1 \longrightarrow BV_c$

- Replace productions of the form $A \longrightarrow B$
  - remove all rules $A \longrightarrow B$
  - For each rule $B \longrightarrow b$ add a rule $A \longrightarrow b$

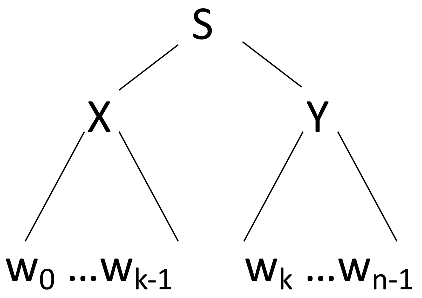
$V_a \longrightarrow a,\ V_b \longrightarrow b,\ V_c \longrightarrow c$

$S \longrightarrow AB,\ A \longrightarrow AV_a \mid a,\ B \longrightarrow V_bV_1 \mid V_bV_c,$

$V_1 \longrightarrow BV_c$

CYK Parsing Algorithm

- Given: Word of length n, $w = w_0w_1\ldots w_{n-1} \in \Sigma^*$
- Case n = 1, i.e., $w = w_0$
  - As grammar is in CNF: Rule $S \longrightarrow w_0$ must exist
- Case n > 1
  - As grammar is in CNF: the word must consist of 2 subwords
  - These can be derived via a production $S \longrightarrow XY$



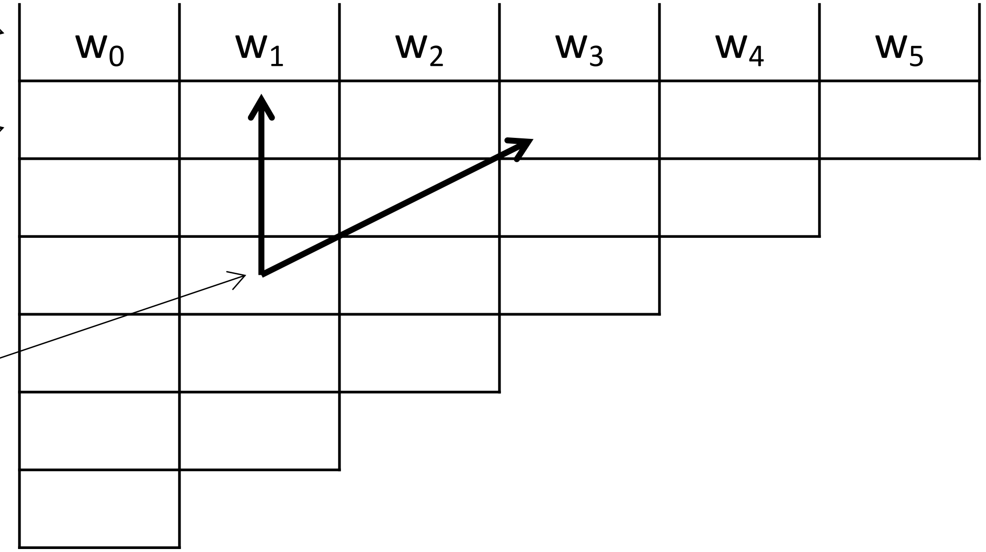
  - The parsing problem has now been reduced to 2 subwords of length k and n – k
  - k is unknown, i.e., we have to look at all possible split positions
  - We now apply the same procedure to both subwords, until we get subwords of length 1

- A table of size n x n is required, but only half of the entries are occupied
- Instead of top-down, CYK operates bottom-up: It starts at subwords of length 1

CYK-Algorithm – Principle

Start with word in top row (not stored in table)

In each field of the 1st row, enter all the variables from which w_i can be generated



For each entry: Check vertically and diagonally to see if there is a rule at the correct distance that generates the subword; if yes: enter the variable(s)

CYK-Algorithm – Principle

What set of variables in the entry T_{ij} generates which subword:

i $\rightarrow$

j $\downarrow$

w_0	w_1	w_2	w_3	w_4	w_5
T_{00}	T_{10}	T_{20}	T_{30}	T_{40}	T_{50}
T_{01}	T_{11}	T_{21}	T_{31}	T_{41}	
T_{02}	T_{12}	T_{22}	T_{32}		
T_{03}	T_{13}	T_{23}			
T_{04}	T_{14}				
T_{05}					

CYK-Algorithm – Principle

Which rules are considered? Search vertically and diagonally, but at the right distance!

i →

j ↓

w_0	w_1	w_2	w_3	w_4	w_5
T_{00}	T_{10}	T_{20}	T_{30}	T_{40}	T_{50}
T_{01}	T_{11}	T_{21}	T_{31}	T_{41}	
T_{02}	T_{12}	T_{22}	T_{32}		
T_{03}	T_{13}	T_{23}			
T_{04}	T_{14}				
T_{05}					

CYK-Algorithm – Principle

Which rules are considered? Search vertically and diagonally, but at the right distance!

i

j

w_0	w_1	w_2	w_3	w_4	w_5
T_{00}	T_{10}	T_{20}	T_{30}	T_{40}	T_{50}
T_{01}	T_{11}	T_{21}	T_{31}	T_{41}	
T_{02}	T_{12}	T_{22}	T_{32}		
T_{03}	T_{13}	T_{23}			
T_{04}	T_{14}				
T_{05}					

CYK-Algorithm – Principle

Which rules are considered? Search vertically and diagonally, but at the right distance!

i →

j ↓

w_0	w_1	w_2	w_3	w_4	w_5
T_{00}	T_{10}	T_{20}	T_{30}	T_{40}	T_{50}
T_{01}	T_{11}	T_{21}	T_{31}	T_{41}	
T_{02}	T_{12}	T_{22}	T_{32}		
T_{03}	T_{13}	T_{23}			
T_{04}	T_{14}				
T_{05}					

CYK-Algorithm – Principle

Which rules are considered? Search vertically and diagonally, but at the right distance!

$i \longrightarrow$

$j \downarrow$

w_0	w_1	w_2	w_3	w_4	w_5
T_{00}	T_{10}	T_{20}	T_{30}	T_{40}	T_{50}
T_{01}	T_{11}	T_{21}	T_{31}	T_{41}	
T_{02}	T_{12}	T_{22}	T_{32}		
T_{03}	T_{13}	T_{23}			
T_{04}	T_{14}				
T_{05}					

CYK-Algorithm – Principle

Which rules are considered? Search vertically and diagonally, but at the right distance!

i

j

w_0	w_1	w_2	w_3	w_4	w_5
T_{00}	T_{10}	T_{20}	T_{30}	T_{40}	T_{50}
T_{01}	T_{11}	T_{21}	T_{31}	T_{41}	
T_{02}	T_{12}	T_{22}	T_{32}		
T_{03}	T_{13}	T_{23}			
T_{04}	T_{14}				
T_{05}					

CYK-Algorithm – Example

- Given: Grammar G in CNF $S \longrightarrow AB$, $A \longrightarrow CD \mid CF$, $B \longrightarrow c \mid EB$
  $C \longrightarrow a$, $D \longrightarrow b$, $E \longrightarrow c$, $F \longrightarrow AD$ start symbol S
- Is w = aaabbbcc $\in$ L(G)?

a	a	a	b	b	b	c	c
C	C	C	D	D	D	B, E	B, E
/	/	A	/	/	/	B	
/	/	F	/	/	/		
/	A	/	/	/			
/	F	/	/				
A	/	/					
S	/						
S							

CYK-Algorithm – Example

$S \longrightarrow AB$, $A \longrightarrow CD \mid CF$, $B \longrightarrow c \mid EB$
$C \longrightarrow a$, $D \longrightarrow b$, $E \longrightarrow c$, $F \longrightarrow AD$

a	a	a	b	b	b	c	c
C	C	C	D	D	D	B, E	B, E
		A				B	

CYK-Algorithm – Example

$S \longrightarrow AB,\ A \longrightarrow CD \mid CF,\ B \longrightarrow c \mid EB$
$C \longrightarrow a,\ D \longrightarrow b,\ E \longrightarrow c,\ F \longrightarrow AD$

a	a	a	b	b	b	c	c
C	C	C	D	D	D	B, E	B, E
		A				B	
		F					

CYK-Algorithm – Example

$S \longrightarrow AB,\ A \longrightarrow CD \mid CF,\ B \longrightarrow c \mid EB$
$C \longrightarrow a,\ D \longrightarrow b,\ E \longrightarrow c,\ F \longrightarrow AD$

a	a	a	b	b	b	c	c
C	C	C	D	D	D	B, E	B, E
		A				B	
		F					
	A						
	F						
A							
S							
S							

Word Problem for Context-sensitive Languages (Type 1)

- Decidable, since the grammar must be monotonic:
  For a word of length n all intermediate results cannot be longer than n symbols
- The number of words with length n over a finite alphabet is finite

Therefore, there must exist an algorithm that solves the word problem:

- Try out all possible derivatives
- Time complexity: $O(a^n)$ with respect to the word length
  - not suitable for practical purposes
  - (a = number of symbols of the alphabet)

Word Problem for Type 0 Languages

- Grammars can have dead-end derivations
  - for type 1 languages it is guaranteed that a dead-end has finite length
  - for type 0 languages a dead-end can also be infinitely long
    (the grammar is not necessarily monotonic)

- The word problem for type 0 languages is unsolvable!
  - There exists no algorithm that can decide for all type 0 languages whether a word w is generated by a given type 0 grammar or not
  - We say the problem is undecidable (*unentscheidbar*)

Application: Compilers

Definition

- Compiler (*Übersetzer*):
  A program that translates the instructions of a program written in one programming language P1 (source language) into instructions of another programming language P2 (target language)
- Source program a $\in$ P1 must be semantically equivalent to target program b $\in$ P2 (they must do the same thing)
- Formal languages are used for this purpose

Types of Compilers

- Compilers in the narrower sense
  - Source language P1 is a higher programming language compared to the target language P2
- Assembler
  - Compiler for transferring assembly language (assembler lang.) source programs to machine language
- Cross-Compiler
  - Compiler generates target code that runs on a different platform than the compiler itself
    - other operating system and/or CPU
- Pre-processor/Pre-compiler
  - Translation of language extensions before actual compilation (typically used in C)
- Compiler-Compiler
  - Program for generating a compiler from a formalized language description
  - e.g.: YACC (Yet Another Compiler Compiler)

Types of Compilers

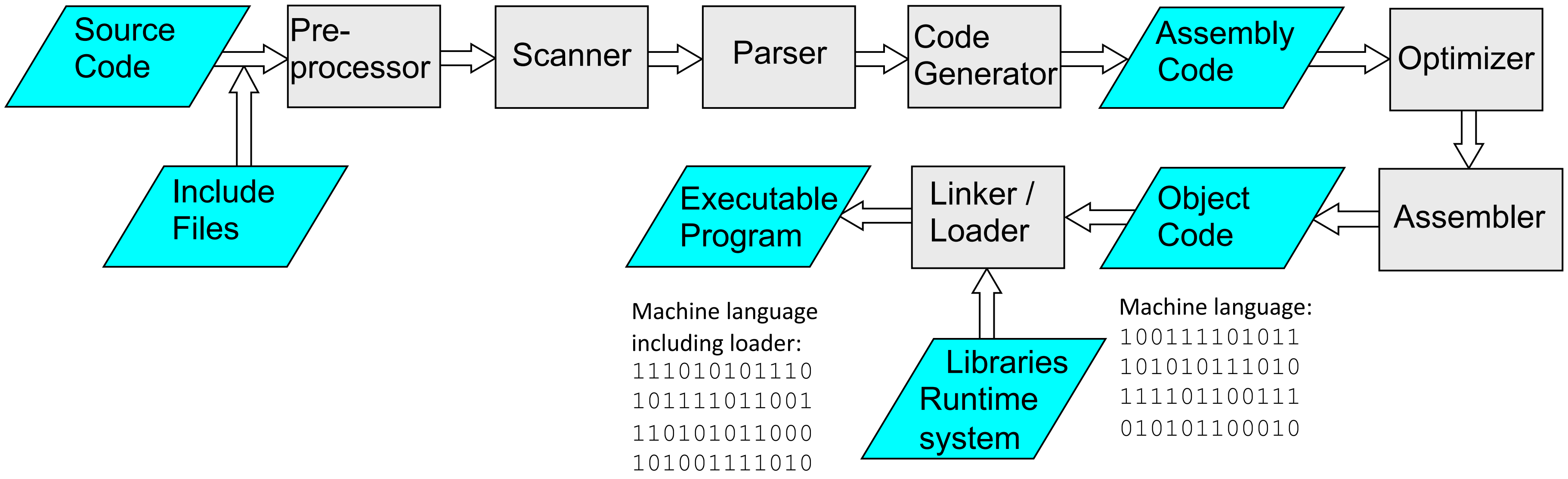
- Interpreter
  - Instructions of the source code are translated and executed immediately
  - Advantage:
    - Tests during development very quick, without a separate compilation step
    - important instrument if program is to run without modification on computers with different operating systems and different hardware
  - Disadvantage: The translation time is always added to the execution time while the program runs
    - costs a lot of time, especially for loops
  - Examples: BASIC, LISP, PROLOG, Python; with restrictions: Java

Compiler: Steps, shown for C Language

```
#include <stdio.h>

int main(void) {
    printf("Hello world\n");
}
```

```
…
movebp, esp
addesp, 204; 000000ccH
cmpebp, esp
…
```



Example: Source Code ⟶ Pre-processor

```
#include <stdio.h>  // Standard Input/ Output  z.B. scanf, printf
#define TEXT "Dies ist ein Text\n"

int main(void)
{
  const float zahl = 5.2f;  // definiert eine Konstante

  printf ("Zahl: %10.2f\n", zahl);  /* Ausgabe */
  printf(TEXT);

  return(0);
}
```

Original source code (complete)

```
…
#line 283 "c:\\program files (x86)\\microsoft visual studio 10.0\\vc\\include\\stdio.h"
 __declspec(dllimport) int __cdecl _pclose(   FILE * _File);
 __declspec(dllimport) FILE * __cdecl _popen(   const char * _Command,    const char * _Mode);
 __declspec(dllimport) int __cdecl printf(    const char * _Format, ...);
…

int main(void)
{
  const float zahl = 5.2f;
  printf ("Zahl: %10.2f\n", zahl);
  printf("Dies ist ein Text\n");

  return(0);
}
```

after pre-processor
(small excerpt)

Example: Assembly Code (Excerpt)

```
_TEXTSEGMENT
_zahl$ = -8; size = 4
_mainPROC; COMDAT
; Line 5
pushebp
movebp, esp
subesp, 204; 000000ccH
pushebx
pushesi
pushedi
leaedi, DWORD PTR [ebp-204]
movecx, 51; 00000033H
moveax, -858993460; ccccccccH
rep stosd
; Line 6
fldDWORD PTR __real@40a66666
fstpDWORD PTR _zahl$[ebp]
; Line 8
fldDWORD PTR _zahl$[ebp]
movesi, esp
subesp, 8
fstpQWORD PTR [esp]
pushOFFSET
??_C@_0O@KIDDCBJA@Zahl?3?5?$CF10?42f?6?$AA@
callDWORD PTR __imp__printf
addesp, 12; 0000000cH
cmpesi, esp
call__RTC_CheckEsp
; Line 9
movesi, esp
pushOFFSET
??_C@_0BD@OPJJLBBI@Dies?5ist?5ein?5Text?6?$AA@
callDWORD PTR __imp__printf
addesp, 4
cmpesi, esp
call__RTC_CheckEsp
; Line 11
xoreax, eax
; Line 12
popedi
popesi
popebx
addesp, 204; 000000ccH
cmpebp, esp
call__RTC_CheckEsp
movesp, ebp
popebp
ret0
_mainENDP
_TEXTENDS
```

Example: Object File (Excerpt)

Line number
(not part of the file)

```
00000f80  02 00 00 00 00 00 87 00  00 00 00 00 00 00 00 00  ................
00000f90  20 00 02 00 2E 72 74 63  24 54 4D 5A 00 00 00 00  .....rtc$TMZ....
00000fa0  08 00 00 00 03 02 04 00  00 00 01 00 00 00 00 00  ................
00000fb0  00 00 03 00 05 00 00 00  CC 11 2E 4C 00 00 00 00  ...........L....
00000fc0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 96 00  ................
00000fd0  00 00 00 00 00 00 08 00  00 00 03 00 00 00 00 00  ................
00000fe0  AD 00 00 00 00 00 00 00  00 00 20 00 02 00 2E 72  ...............r
00000ff0  74 63 24 49 4D 5A 00 00  00 00 09 00 00 00 03 02  tc$IMZ..........
00001000  04 00 00 00 01 00 00 00  00 00 00 00 03 00 05 00  ................
00001010  00 00 9E 7A 90 5D 00 00  00 00 00 00 00 00 00 00  ...z.]..........
00001020  00 00 00 00 00 00 00 00  BC 00 00 00 00 00 00 00  ................
00001030  09 00 00 00 03 00 00 00  00 00 D3 00 00 00 00 00  ................
00001040  00 00 00 00 20 00 02 00  2E 64 65 62 75 67 24 54  .........debug$T
00001050  00 00 00 00 0A 00 00 00  03 02 74 00 00 00 00 00  ..........t.....
00001060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00001070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 E2 00  ................
00001080  00 00 3F 3F 5F 43 40 5F  30 42 44 40 4F 50 4A 4A  ..??_C@_0BD@OPJJ
00001090  4C 42 42 49 40 44 69 65  73 3F 35 69 73 74 3F 35  LBBI@Dies?5ist?5
000010a0  65 69 6E 3F 35 54 65 78  74 3F 36 3F 24 41 41 40  ein?5Text?6?$AA@
000010b0  00 5F 5F 69 6D 70 5F 5F  70 72 69 6E 74 66 00 3F  .__imp__printf.?
000010c0  3F 5F 43 40 5F 30 4F 40  4B 49 44 44 43 42 4A 41  ?_C@_0O@KIDDCBJA
000010d0  40 5A 61 68 6C 3F 33 3F  35 3F 24 43 46 31 30 3F  @Zahl?3?5?$CF10?
000010e0  34 32 66 3F 36 3F 24 41  41 40 00 5F 5F 72 65 61  42f?6?$AA@.__rea
000010f0  6C 40 34 30 61 36 36 36  36 36 00 5F 5F 66 6C 74  l@40a66666.__flt
00001100  75 73 65 64 00 5F 5F 52  54 43 5F 43 68 65 63 6B  used.__RTC_Check
00001110  45 73 70 00 5F 5F 52 54  43 5F 53 68 75 74 64 6F  Esp.__RTC_Shutdo
00001120  77 6E 2E 72 74 63 24 54  4D 5A 00 5F 5F 52 54 43  wn.rtc$TMZ.__RTC
00001130  5F 53 68 75 74 64 6F 77  6E 00 5F 5F 52 54 43 5F  _Shutdown.__RTC_
00001140  49 6E 69 74 42 61 73 65  2E 72 74 63 24 49 4D 5A  InitBase.rtc$IMZ
00001150  00 5F 5F 52 54 43 5F 49  6E 69 74 42 61 73 65 00  .__RTC_InitBase.
```

Machine code
(hexadecimal)

Interpretation
as ASCII

Example: Executable Program (Excerpt)


```
0000000000000000000000000000000000000¢ppN0000B000j000ÀcA0ÀM00Dies
 ist ein TextJ000000Zahl: %10.2fJ000ff¦@f0:0\0d0d0\0v0c0t0o0o0l0
s0\0c0r0t0_0b0l0d0\0s0e0l0f0_0x08060\0c0r0t0\0s0r0c0\0c0r0t0e0x0
e0.0c0000000000000000000000000000_0_0n0a0t0i0v0e0_0s0t0a0r0t0u0p0

 ...

00000000D00000A0X000X00€00000000D00000A0A000000€00000000D00000A0
>D00H000X A0ZA00äD000000<assembly xmlns="urn:schemas-microsoft-c
om:asm.v1" manifestVersion="1.0">MJ  <trustInfo xmlns="urn:schem
as-microsoft-com:asm.v3">MJ    <security>MJ      <requestedPrivi
leges>MJ        <requestedExecutionLevel level="asInvoker" uiAcc
ess="false"></requestedExecutionLevel>MJ      </requestedPrivile
ges>MJ    </security>MJ  </trustInfo>MJ</assembly>PAPADDINGXXPAD
DINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPAD

0PA0Ø000 3³3¹3Ê3Ð3T4"656‹6‟6ž6£6¨6-6³6¹6À6Æ6Ø6à6æ6ô6ÿ667C7H7M7S7
Y7^7c7i7q7w7È7Í7ß7÷7H83898^8f8{8„8 8’8¼8Æ8Î8Ó8á8ë8ó8ÿ8G9]9#9)919
I9Q9Z9`9g9n9u9ƒ9‰9‘9˜9ž9§9Ö9Ü9å9ì9ò9û9G:Û:p;w;’;Ü;+<r<-<Ñ<[=;=`=
‘=¹=x>}> >Î>(?-??? ?ü?000 A0ð000G0R0Œ0À0Å0<1H1d1j1w1~1ƒ1ž1¤1-1¶1
»1Æ1æ1ï1ø1E2S2B2ì2ó2L3W3’3Ì3O4E4 4³4°4Ù4ß4å4>5O5U515A5H5Ö5Ü5ì5ú5
ÿ5D6t6¤6«6¾6Ô6Û6î6H7M7¶737:7M7l7s7{7,7—7-7²7¼7Á7V8\8b8^8“8_8¨8·8
Ì8Ø8ä8ô8=9H9Š9 9-9œ9¢9Ø:Ý:ï:H;(<.<4<V<[<§<Ä< =)=6=@=H=M=T=¥=s>z>
ü>C?ü?0000A0¼000G0!0&0,0<0Z0_0,0 0©0¿0ô0S1x1~1 1-1Ÿ1¨1²1Æ1b2t2z2

€2†2Œ2’2˜2¬2²2¸2¾2Ä2Ê2Ñ2Ø2ß2æ2í2ô2û2C3K3S3¶3(3-333=3G3S3_3d3v3{3
 3‡3Ÿ3¦3ô3ú3O4F4L4R4X4^4$4*40464<4B4H4N4T4Z4`4f4l4r4x4~4„4Š4 4-4
œ4000PA0 000D1P4T5d;h;l;p;t;x;ø?ü?000`A0(00000D0H0„2^2D6P9l;p;Œ;
 ;¬;°;Ð;ì;ð;000000000000000000000000000000000000000000000000000000
```

Scanner/Lexical Analysis

- Lexical analysis (*lexikalische Analyse*)
  - Conversion of the source program a $\in$ P1 with scanner into intermediate code (token)
  - Objects of the language (e.g., operators, keywords, identifiers) are recognized as such and converted into tokens
  - Simple rule violations can already be reported here
    - e.g., use of an illegal character in an identifier
  - Description by regular grammar/regular expressions
  - Implemented as deterministic finite automaton

Parser/Syntactic & Semantic Analysis

- Syntactic analysis (syntakt*ische Analyse*)
  - Parser generates the syntax tree of the program a∈P1 from tokens according to the syntax of P1
  - Uses deterministic context-free grammars
    - Top-Down: LL(k) grammar, usually LL(1)
    - Bottom-Up: LR(k) grammar, usually LR(1)
  - Implemented as deterministic pushdown automaton

- Semantic analysis (*semantische Analyse*)
  - Analysis of the syntax tree of a∈P1: Check the semantics of the program, e.g.,
    - Have all variables used been declared?
    - Are they used according to their type?
    - Are there violations of range limits?
  - Simultaneously: Code generator transfers a to the target language P2
  - Result: Target program b∈P2
  - Uses (context-free) attribute grammars (*Attributgrammatiken*)

Code Optimization

Technische Hochschule Rosenheim

- Code Optimization
  - Goal: Increase efficiency of the target program $b \in P2$
  - Optimize runtime and/or memory requirements; usually this is a trade-off
  - Code optimization is time-consuming
  - Program code $b \in P2$ is modified
    - Semantics, of course, should remain unchanged
    - Problem: Complete preservation of the semantics of b cannot be guaranteed in every case
  - Therefore: optional step, with various degrees of more or less aggressive optimization

Tools

- lex / flex: Lexical analysis
  - lex: 1975
- yacc / bison: syntactic/semantic analysis
  - yacc: 1979
- Generate C code files
- Links:
  - lex & yacc: http://dinosaur.compilertools.net/
  - flex: http://flex.sourceforge.net/
  - bison: http://www.gnu.org/software/bison/

"The asteroid to kill this dinosaur is still in orbit."
(Lex Manual Page)

Summary

- Word problem
  - regular language: finite automaton
  - context-free language: CYK algorithm
  - context-sensitive language: brute-force – try out all possibilities
  - Type 0 language: no algorithm exists (undecidable)
- Main steps during compilation of a program
  - lexical analysis
    - scanner
    - conversion to tokens
    - regular grammar/DFA
  - syntactic analysis
    - parser
    - generation of a syntax tree
    - deterministic context-free grammar/DPDA
  - semantic analysis
    - Analysis of syntax tree
    - context-free attribute grammars
  - Code generation and optimization
  - Linking

Sources

- H. Ernst, J. Schmidt und G. Beneken: *Grundkurs Informatik*. Springer Vieweg, 7. Aufl., 2020.
- Schöning, U.: *Theoretische Informatik – kurz gefasst*. Spektrum Akad. Verlag (2008)
- Sander P., Stucky W., Herschel, R.: *Automaten, Sprachen, Berechenbarkeit*, B.G. Teubner, 1992