



Context-Free Language Translation for the DDC Assembler

Jim Gaines



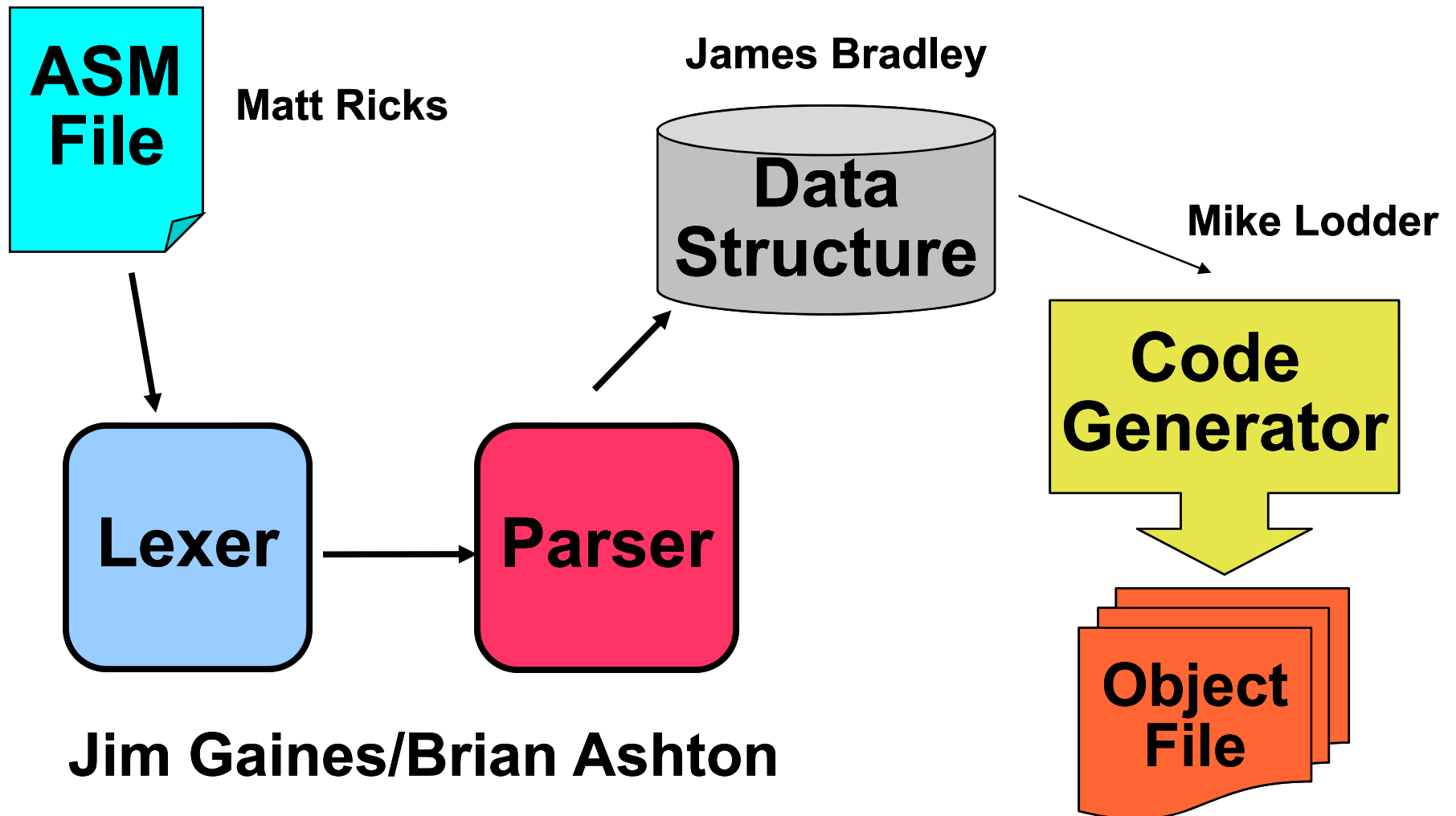
DDC Language Translation



- **Assembler front end (analysis phase)**
- **Interface between developer and image creation**
- **Lexical (lexer) and syntactic (parser) analysis**
- **Akin to word processor spelling/grammar checks**
- **Method calls to backend for object creation (synthesis phase)**



Overview





DDC Language Translation



- **Problem:**
 - **Check source for correctness**
- **Solution:**
 - **Develop a set of spelling (lexical) rules**
 - **Develop a set of grammatical (syntactic) rules**
 - **Design tools that use rules to check for errors (lexer/parser)**



Source Analysis



- **FOO not recognized**

XEQ LT 0x0FFB

correct

FOO LT 0x0FFB

exception

- **Opcode can not follow an opcode**

JMP NO_FMT_ERR 0x13F4

correct

JMP JMP 0x13F4

exception



Language Ambiguities



- **“He was mad” (angry or insane?)**
- **“The lady scientist made the robot fast while she ate” (at least 4 meanings!) [1]**
- **Ambiguities can not be present in translation from source to object code**

[1] John R. Pierce, “An Introduction to Information Theory”, Dover Publications, New York (1980)



Context-Free Grammar



- **Consists of four components [2,3]:**
 - **A set of tokens, known as *terminal* (keyword) symbols.**
 - **A set of nonterminals.**
 - **A set of productions, i.e. a nonterminal followed by a sequence of tokens and/or nonterminals**
 - **A designation of one of the nonterminals as the *start* symbol.**

[2] Noam Chomsky, “Syntactic Structures”, Mouton and Co., 1957

[3] A.V. Aho, et. al., “Compilers, Principles, Techniques and Tools”, Addison Wesley (1986)



Lexer/Parser Design



- **Notoriously difficult to design**
 - Often considered a “black art”
- **Challenge to debug**
 - Hand written lexer/parser may not be intuitive
- **Not easily extended**
 - Inertial to future changes in language



Lexer/Parser Generators



- **Backus-Naur Format (BNF) used to define language specification**
- **Generates optimized lexer/parser from formal language definition**
 - **Input: language definition (in BNF)**
 - **Output: lexer and parser**
- **e.g. GNU Flex/Bison and Lex/Yacc**



ANTLR: A Predicated-LL(k) Parser Generator



- Large support base, active website/mailling list (www.antlr.org)
- ~20 years of development, result of PhD thesis
- Becoming universal (>5k downloads/month)
- Uses an extension of Backus-Naur Format (EBNF)
- Similar to GNU Flex/Bison and Lex/Yacc



ANTLR: A Predicated-LL(k) Parser Generator



- **Open source and license free**
- **Hierarchical exception handling**
- **Syntactic predicates**
 - **Temporarily set $k=\infty$ (details given later)**
- **Novel approach: define lexer/parser in EBNF**
- **Very rich, powerful and complex package**



DDC Lexer Grammar



<SUM> ::= +

<DIF> ::= -

<MUL> ::= *

<DIV> ::= /

...

<NUMBER > ::= <DEC> | <HEX> | <OCT> | <BIN>

<DEC> ::= (0 .. 9)+ //decimal

<HEX> ::= 0x (0 .. 9 | a .. f | A .. F)+ //hexadecimal

<OCT> ::= 0o (0 .. 7)+ //octal

<BIN> ::= 0b (0 | 1)+ //binary

COMMENTS ::= // (~'\n')*

(continued..)



DDC Lexer Grammar



(continued..)

<WHITE_SPACE> ::= ' ' | \t | \n | \r

<CHARACTER> ::= ' . ' //any character in single quotes

<STRING> ::= " (.) * " //zero or more characters in double quotes

<IDENT> ::= (a .. z | A .. Z) (a .. z | A .. Z | 0 .. 9 | _) *



ANTLR Generated Lexer



```
switch ( cached_LA1 ) {
  case ' \t ' : case ' \n ' : case ' \r ' : case ' ' : {
    mWS(true);
  }

  case ' + ' : {
    mSUM(true);
  }

  case ' - ' : {
    mSUM(true);
  }

  ...
}
```



Token Stream



- If character stream passes lexical analysis, a token is created and passed to parser
- [“ID”, <type>, line #, column #]
 - e.g. [“ORG”, <6>, line 3, column 1]
- Lexer analyzes a stream of characters, parser analyzes a stream of tokens



DDC Parser Grammar



<program> ::= ((IDENT CLN)? instr)+

<instr> ::= opcode_u expr | opcode_cp cond expr | opcode_cnp cond

//keywords

<opcode_cp> ::= XEQ | IRQ | DLY | LTT | LFT | PSI | PSM | JMP | CAL

<opcode_cnp> ::= RTN | HLT | WFT | SFT | PTT | PBS | WTG

<opcode_u> ::= CFT | CMT | FLG | XQF | ORG | EQU | FCB | ...

<cond> ::= LT | GT | EQ | NEQ | GP0_0 | GP0_1 | GP1_0 | ...

(continued..)



DDC Parser Grammar



(continued..)

<expr> ::= mexpr (SUM mexpr | DIF mexpr)*

<mexpr> ::= atom (MUL atom | DIV atom | MOD atom)*

<atom> ::= NUMBER | IDENT | LPARN expr RPARN

//note the recursiveness – this allows for expressions like (3+4)*2+label/3



ANTLR Generated Parser



```
public void program() { /*throws RecognitionException,  
                        TokenStreamException*/  
    for (;;) {  
        if (token_exists) {  
            instr();  
        }  
        else {  
            throw new NoViableAltException(LT(1), getFilename());  
        }  
    }  
    return;  
}
```



Lexer/Parser Objects in C#



- *.asm passed as CharBuffer object to lexer
- Lexer passed to parser
- Parser calls program(), the grammar start symbol

```
17     public static void Main(string[] args){
18         L lexer = new L(new CharBuffer(new StreamReader("test.asm")));
19         P parser = new P(lexer);
20
21         try{
22             parser.program();
23             Console.ReadLine(); //Pause
24         }
25         catch (Exception e){
26             Console.Error.WriteLine("problem with stream: "+e);
27             Console.Error.WriteLine(e.StackTrace);
28         }
29     }
```



Questions?



-
- [1] John R. Pierce, “An Introduction to Information Theory”, Dover Publications, New York (1980)
 - [2] Noam Chomsky, “Syntactic Structures”, Mouton and Co., 1957
 - [3] A.V. Aho, et. al., “Compilers, Principles, Techniques and Tools”, Addison Wesley (1986)
 - [4] T.J. Parr, R.W. Quong, “ANTLR: A Predicated-LL(k) Parser Generator”, *Software--Practice and Experience*, 25 no. 7, pp 789–810, 1995