Yacking About Bison

James K. Lowden
Cobolworx

Symas Corporation

Brussels, 4 February 2024 11:20, k4201

Why COBOL?
A COBOL compiler?
- Learning to parse
- Learning to parse COBOL

Bison Lessons, and a challenge

>

COBOL

Problem statement

Big software for big businesses

- Billions of lines still in use
    - Banks, Insurance, Railroads
- Vital, line-of-business applications
    - Fast: designed for small computers
    - Complex: decades of compressed spaghetti
- Mostly written in 1970s and 1980s ...
- ... for systems that no longer exist

100% proprietary software

- OS
- Compiler, etc.
- File system, DBMS, Scheduler
- Security

COBOL

Business Proposition

Expensive licensing
Lift and Shift only feasible migration path
GCC COBOL targets ISO 1985-2023

- *Batteries not included*
- Tailor to proprietary requirements

COBOL

*CO*mmon *B*usiness *O*riented *L*anguage

Defined 1957
- Backus–Naur form first used 3 years later, for ALGOL 60

Not Computer Science
- No standard library
- No functions
- No recursion (until circa 1985)

Compiler does everything
- Type conversion (via `MOVE`)
- Record-oriented I/O
- Fast

A Big Language

How big is “big” ?

C:	32 keywords
ISO COBOL:	371 keywords, including 51 verbs
GCC COBOL:	661 terminals, 1661 rules

CALL statement (format 1)

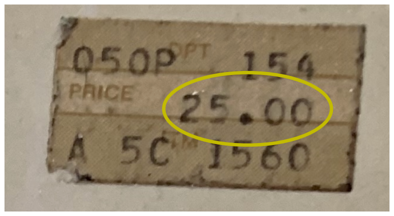
$$\mathit{CALL} \left\{\begin{array}{l} \mathit{identifier\text{-}1} \\ \mathit{literal\text{–}1} \end{array}\right\} \left[\mathit{USING} \left\{\begin{array}{l} [\mathit{BY\ REFERENCE}]\ \{\mathit{identifier\text{-}2}\}\cdots \\ \ \mathit{BY\ CONTENT}\ \{\mathit{identifier\text{-}2}\}\cdots \end{array}\right\}\cdots\right]$$

$$[\mathit{RETURNING\ identifier\text{-}3}]$$

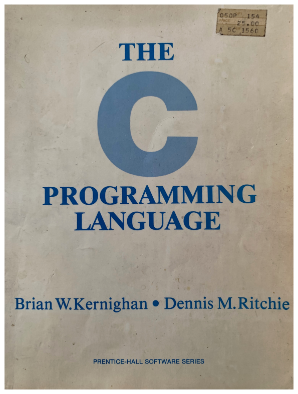
$$\left[\begin{array}{l} \mathit{ON\ EXCEPTION\ imperative\text{-}statement\text{-}1} \\ \mathit{NOT\ ON\ EXCEPTION\ imperative\text{-}statement\text{-}2} \end{array}\right] [\mathit{END\text{-}CALL}]$$

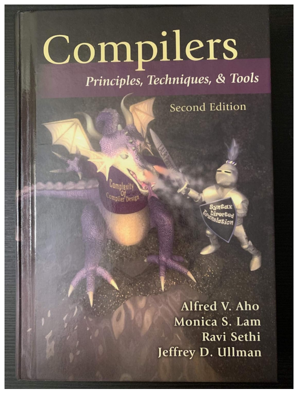
Parsing

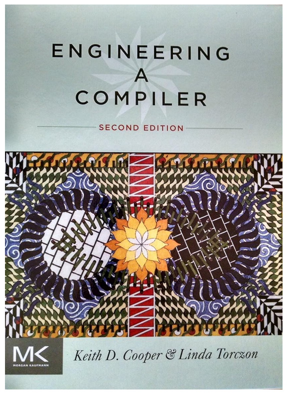
Enter the Parser



Just compile the compiler, right?


THE C PROGRAMMING LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
PRENTICE-HALL SOFTWARE SERIES





Parsing, Parsing

Computer Science: how to write a compiler
My job: write a parser for GCC

- Textbooks cover theory, not practice
- LALR(1) appears in a small section of each book

Question: *How do I parse COBOL?*
Answer: *A handle of a right-sentential form Υ is a production $A \rightarrow B$ and a position of Υ where the string B may found, such that replacing B at that position by A produces the previous right-sentential form.*

Parsing takes practice

Learning to love your parser generator

No royal road to writing a parser
All problems simple, in theory
Problem is concrete, solution is abstract
Small community

- help-bison@gnu.org had 0 messages in January 2024
- especially of COBOL (compiler) experts

Everyday needs not obvious to beginner

Flex & Bison

The oil and vinegar of free software

2 separate projects that share common variables

- Really?
- **flex** says it needs `%option bison-bridge` but I don't use it
- **bison** pretends **flex** does not exist
  - ☞ *The "lexical analyzer" function, 'yylex', recognizes tokens.... The function is sometimes referred to as a lexical scanner.*

Bison, the Enigma

What no one tells you

Two levels of C

Rule C is not Action C

nonterminals are precedence

Much of precedence is determined by what reduces what

Tracing is all

Traces show what the grammar does

types, types, types

The more tokens, the easier to distinguish in the grammar

Bison, the Enigma

Dead ends

`%prec`

- Useful for `%empty`
- No luck with `AND/OR`

`-Wcounterexamples`

Only demonstrates what is proved

`--graph` *for large grammar*

- Bison dies trying for `gcobol`
- Probably inscrutable at scale, anyway

Bison, the Hydra

Options, options everywhere

Compatibility is hard

```
// Use %defines instead of %header for Bison 3.5.2 compatibility.
%defines "parse.h"
```

TIMTOWTDI: command line versus directives

So many things not to use

- Pure parser
- Push parser
- C++ parser
- GLR, IELR(1), and LR(1) parsers
- `yacc` emulation

Bison, the attic

Complex important features

`%code`

top	`requires` *should usually be more appropriate*
requires	*best place to define types referenced in* `%union` *directives*
provides	*definitions and declarations [for] other modules*

`%locations`

associates a line number with every token

Bison, reorganized

prototypical metafile

```
%code requires {
#include "parse.requires.h"
}
%code provides {
#include "parse.provides.h"
}
%{
 // static parser-local function declarations
%}
%union {
 // types
}
// %printer, 1 per type
%%
%%
#include "parse.post.h" // static definitions
```

- Editor not confused by “yacc mode”
- Clear where to put semantic types

Bison, pure gold

Are you my type?

%union	best way to define semantic types
%printer	every type needs string representation

```
Reducing stack by rule 1000 (line 6146):
   $1 = token VARYING (15.1: )
   $2 = nterm num_operand (15.1: FldNumericDisplay WCCSP-WORK-SUB)
   $3 = token FROM (15.1: )
   $4 = nterm num_operand (15.1: FldLiteralN _stack3)
   $5 = nterm vary_by (15.1: FldLiteralN _stack4)
   $6 = nterm perform_cond (16.1-18.0: FldConditional _stack5 )
-> $$ = nterm varying (15.1-18.0: )
```

The Way of the Bison

think, think, think

Declarative languages require logical debugging

- Pointless to debug generated code
- Rules rule!
- Traces show the LALR table in action
- `%verbose` output explains trace

Bison versus COBOL

audience challenge

COBOL defines *Complex Abbreviated Relation Conditions* meaning e.g.

```
A = B OR C AND < D
```

expands to

```
A = B OR (A = C AND A < D)
```

If you know the right way to solve that, I am jklowden@acm.org.