Introducing COBOLOG

The **Co**mmon **B**usiness-**O**riented **Log**ic-programming language

Michael Coulombe

Electrical Engineering and Computer Science Department
Massachusetts Institute of Technology
32 Vasser Street, Cambridge, MA

mcoulomb@mit.edu

Abstract

We introduce COBOLOG, a new cross-paradigm programming language. Inspired by two popular languages, COBOL and PROLOG, we finally bring logic programming to the business community.

1. Introduction

COBOL, the Common Business Oriented Language, was released in 1961 to much excitement as "COBOL 61" [17]. After a French grad student named Alain Colmerauer was tasked to write a syntax-directed error detector for COBOL programs, he fled to Canada and created PROLOG, "PROgrammation en LOGique" [7]. Computer scientists have been split between academia and industry ever since.

We celebrate the 60th anniversery of COBOL 61 by reunifying these two great languages, combining the best of the business-oriented and logic-oriented programming paradigms for the first time in COBOLOG. In this paper, we will detail our new language and document the current progress on its implementation.

1.1 Mandatory COBOL Acknowledgements

This publication is based on the COBOL System developed in 1959 by a committee composed of government users and computer manufacturers. The organizations participating in the original development were:

- *Air Materiel Command, United States Air Force*
- *National Bureau of Standards, U. S. Department of Commerce*
- *Burroughs Corporation*
- *David Taylor Model Basin, Bureau of Ships, U. S. Navy*
- *Electronic Data Processing Division, Minneapolis-Hoimywell Regulator Company*
- *International Business Machines Corporation*
- *Radio Corporation of America*
- *Sylvania Electric Products, Inc.*
- *Univac Division of Sperry-Rand Corporation*

In addition to the organizations listed above, the following other organizations participated in the work of the Maintenance Group:

- *Allstate Insurance Company*
- *Bendix Corporation, Computer Division*
- *Control Data Corporation*
- *Dupont Corporation*
- *General Electric Company*
- *General Motors Corporation*
- *Lockheed Aircraft Corporation*
- *National Cash Register Company*
- *Philco Corporation*
- *Standard Oil Company (N. J.)*
- *United States Steel Corporation*

This COBOL-61 manual is the result of contributions made by all of the above-mentioned organizations. No warranty, expressed or implied, is made by any contributor or by the committee as to the accuracy and functioning of the programing system and language. Moreover, no responsibility is assumed by any contributor, or by the cominittee, in connection therewith.

It is reasonable to assume that a number of improvements and additions will be made to COBOL. Every effort will be made to insure that the improvements and corrections will be made in an orderly fashion, with due recognition of existing users' investments in programing. However, t h i s protection can be positively assured only by individual implementors.

Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures and the methods for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

The authors and copyright holders of the copyrighted material used herein: FLOW-MATIC (Trade-mark of Sperry-Rand Corporation), Programing for the UNIVAC® I and II, Data Automation Systems© 1958, 1959, Sperry-Rand Corporation; IBM Commercial Translator, Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell, have specifically authorized the use of this material, in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programing manuals or similar publications.

Any organization interested in reproducing the COBOL report and initial specifications in whole or in part, using ideas taken from this report or utilizing this report as the basis for an instruction manual or any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention "COBOL" in acknowledgment of the source, but need not quote this entire section.

1.2 Background

COBOL was known as a difficult language from its inception, and even today mere mortals have trouble distinguishing their perceptions from the reality hiding behind their terminal [5]. Declassified documents show that in the 80's, even DoD researchers could only semi-automatically translate a subset of business programs into human-understandable language [10]. Some say COBOL was prophesized in 1928 by Hewlett-Packard Lovecraft before he started his successful technology company, which went on to develop its own COBOL compiler [19]:

> *The most merciful thing in the world, I think, is the inability of the human mind to correlate all its contents. We live on a placid island of ignorance in the midst of black seas of infinity, and it was not meant that we should voyage far. The sciences, each straining in its own direction, have hitherto harmed us little; but some day the piecing together of dissociated knowledge will open up such terrifying vistas of reality, and of our frightful position therein, that we shall either go mad from the revelation or flee from the deadly light into the peace and safety of a new dark age.*
>
> — HP Lovecraft, The Call of COBOL [14]

Despite the success of COBOL, it was unable to stop the rise of PROLOG. Its first interpretor was written for the sake of portability in FORTRAN, a unfortunate necessity of the time [7]. 80's researchers could see the value in this language, but experiments with concurrently teaching PROLOG and PASCAL to "Business Information Systems" majors in universities [12] just confused the poor students greatly and left them unprepared for COBOL on the job.

In the wake of a world-wide pandemic, we believe time is ripe to loop back and disrupt the computer science community, giving 110% for all the stakeholders by leveraging our core competency: creating new programming languages.

2. The COBOLOG Language

COBOLOG, short for the **Com**mon **B**usiness-**O**riented **Log**ic-programming language, combines a variety of aspects from COBOL and PROLOG, mixed with a tablespoon of modern innovations.

2.1 Syntax

A COBOLOG program is a collection of logical predicates on terms and relations between them. Relations are defined by a set of clauses, typed like `Pred :- Goal`, which states that proving `Goal` is true is one way to prove `Pred` is true. A term is one of:

- Compound, a bunch of words each starting with lower-case letters and possibly interspersed with other terms
- Variable, a bunch of words starting with upper-case letters
- Integer, written as contiguous base 10 digits
- Operator expression, either nullary, prefix, infix, or postfix
- List, including finite (`[4,2,0]`), cyclical (`L is [go|L]`), and list prefixes with non-list tails (`[A,B|666]`)
- Indented block, each line containing a goal (predicate term)

Here in the middle of the page is an example COBOLOG program:

```
main :-                             % this predicate is the one your computer will start trying to prove
    the sum of [1,2,3,4] is The Sum % performs the predicate defined below
    display The Sum                 % prints the sum, which is 10, just like in COBOL
the sum of [] is 0                                                % an empty list has sum zero
the sum of [The First Element|The Rest Of The List] is The Sum :- % a non-empty list is harder to check
    the sum of The Rest Of The List is The Sum Of The Rest        % performs this predicate recursively
    add The First Element to The Sum Of The Rest giving The Sum   % adds two numbers, like in COBOL
```

So we are all on the same page (page 2), below is an enterprise next-gen COBOL program for rockstars that computes the sum.

```
*>****************************
*> Ph'nglui mglw'nafh COBOL **
*> R'lyeh wgah'nagl fhtagn  **
*>****************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.SUM.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 LIST          PIC 9  OCCURS 4 TIMES.
 01 LIST-LENGTH   PIC 9  VALUE 4.
 01 CUR-SUM       PIC 99 VALUE 0.
 01 CUR-POS       PIC 9  VALUE 1.
 PROCEDURE DIVISION.
 000-MAIN SECTION.
  MOVE 1 TO LIST(1).
  MOVE 2 TO LIST(2).
  MOVE 3 TO LIST(3).
  MOVE 4 TO LIST(4).
  PERFORM VARYING CUR-POS FROM 1 BY 1
    UNTIL CUR-POS IS GREATER THAN LIST-LENGTH
   ADD LIST(CUR-POS) TO CUR-SUM
  END-PERFORM.
  DISPLAY CUR-SUM.
```

The above business-ready, game-changing COBOLOG program proves that the list `[1,2,3,4]` has a sum then prints it to the screen for customer-centric convenience. It uses recursion, a feature not present in COBOL 61 that we will cover in Section 3.

We adopt a holistic approach with modern preferences ala PYTHON and others by writing the conjunction of goals in indented blocks (spaces only, of course) without line- or block-ending delimeters. We can start a line with the "\" character to continue the previous goal in the same block.

Next, here is a premium free trial sneak peak of the conditional predicate `if C then T else F`. Currently, in COBOLOG, nested goals must be written in a block to empower your computer to prove them, helping you keep your ducks in a row.

```
% The Law of Identity (this is not legal advice)
A equals A
% Relating a number to its sign {-1,+1}
Number has sign Sign :-
    if
        % if there are multiple solutions here,
        % only the first one will be used
        Number is less than 0
    \ then
        add Sign to 1 giving 0
    \ else
        Sign equals 1
```

3. Implementation

The tools we use have a profound (and devious!) influence on our thinking habits, and, therefore, on our thinking abilities.

— Dijkstra, COBOL enthusiast and noted `goto` critic [9]

The COBOLOG Synergizer™ is a fast-paced innovative tool that processes your COBOLOG code into executable content for your computer. In this section, we will give a 30,000-foot view of the current implementations of the Synergizer™ and the all-new Teamworker™ run-time system for executing your COBOLOG programs.

3.1 Synergizer™

The Synergizer™ is a transpiler written in PROLOG. It is based on the dragon method, being the composition of four stages: lexing, parsing, semantic analysis, and code generation [4]. The toolchain is built on military-grade industry-proven machine learning block chain algorithms like Pratt Parsing [16].

As a logic program, the Synergizer™ has the never-before-seen feature that it can also uncompile your executable content back into source code, saving your company billions of dollars in unnecessary storage space.

The output of the Synergizer™ is a file containing source code in the D programming language, the second-best language after COBOLOG (PROLOG is third, then COBOL next, obviously), for input into the Teamworker™.

3.2 Teamworker™

We considered a variety of logic programming implementation techniques [13, 20], but decided to pivot away, get in the weeds, drill down, think outside the box, lean in, take it offline for a moment, be a thought leader, and roll our own data-driven scalable best-of-breed innovative system, the Teamworker™.

At the end of the day, we exploit the most cloud-ready feature of D, ranges, to implement a list monad. Predicates are roughly transpiled as closures over their parameters that take the state-of-the-world as an argument and return a range over new states. We avoid the pain points by using exciting new algorithms for persistant memory [8] and the unification of arbitrary cyclic terms [11] with great ROI.

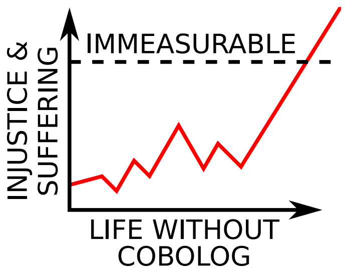
Another reason we chose D was to improving upon COBOL's lack of recursion. D includes a run-time stack that automatically pushes and pops return addresses from function calls, like some other modern languages. We also rely heavily on D's bleeding-edge garbage collector to deal with the many allocations that COBOLOG users can rely on for their organic omni-channel win-win DevOps mindfulness.

3.3 Experimental Results

We ran the Synergizer™ on the summing program on the previous page, using the SWI-Prolog (threaded, 64 bits, version 8.2.3), then compiled the output with the DMD64 D Compiler v2.095.0, then ran the Teamworker™ executable content on a four-core Intel®Core™ i5-4300M CPU @ 2.60GHz running Ubuntu 20.04.2 LTS to obtain these amazing performance results:

SWI-Prolog	DMD	Intel
0.012s (56,348 inferences)	1.417s	0.002s

Wow!
Amazing!



4. Testimonials

As you can see from the graph above, we are proud to provide an actionable and entergaging solution to some of the most troubled generations of computer scientists and business people. If you aren't convinced yet, here are some quotes from just a few of the many happy users of COBOLOG:

I used to think recursion was overrated – the 'banana slicer' of the programming world; a tool used by pretentious graduate students with more time than sense. In COBOLOG, *however, recursion reaches its apex. I cannot recommend* COBOLOG *too highly.*

— World famous language connoisseur and noted kitchen gadget critic [3]

Paying the bills each month was a total chore. But once I started using COBOLOG, *paying bills became so easy, like I didn't have to do anything at all! And I have so much more . . . Oops sorry, the lights just went out.*

— The accountant of Dewey Stealfromem and Howe Inc. and noted thinking critic [2]

Why can't I just eat my waffle?

— Barack Obama, noted Bubble Sort critic [6, 18]

If I had been able to use COBOLOG *during my working career, not only would I have been able to more easily sell more people more products possessing fewer benefits, but my hair would have been silkier and more manageable. Its impact on our understanding of toothbrushes alone is impossible to overstate. It is truly THE turning point in human history.*

— Retired advertising copywriter and noted refrigerator magnet critic [1]

5. Conclusion

Ask your doctor of computer science if COBOLOG is right for your business, or just email us with the details of your corporate needs to receive a quote today. Business moves fast, especially in the software industry, so I leave you with these wise words:

Every step of real movement is more important than a dozen programs.

— Karl Marx, noted program critic [15]

References

[1] Anonymous retired advertising copywriter and noted refrigerator magnet critic. Personal communication, Mar. 2021.

[2] Anonymous accountant of dewey stealfromem and howe inc. and noted thinking critic. Personal communication, Mar. 2021.

[3] Anonymous world famous language connoisseur and noted kitchen gadget critic. Personal communication, Feb. 2021.

[4] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers, principles, techniques, and tools*. Addison-Wesley Pub. Co, Reading, Mass, 1986. ISBN 978-0-201-10088-4.

[5] E. Arranga and F. Coyle. Cobol: perception and reality. *Computer*, 30(3):126–128, Mar. 1997. ISSN 00189162. doi: 10.1109/2.573683. URL http://ieeexplore.ieee.org/document/573683/.

[6] C. Bateman. Obama's $10,000 waffle, 2008. URL https://www.vanityfair.com/news/2008/04/obamas-10000-wa.

[7] A. Colmerauer and P. Roussel. The birth of Prolog. In T. J. Bergin and R. G. Gibson, editors, *History of programming languages—II*, pages 331–367. ACM, New York, NY, USA, Jan. 1996. ISBN 978-0-201-89502-5. doi: 10.1145/234286.1057820. URL http://dl.acm.org/doi/10.1145/234286.1057820.

[8] S. Conchon and J.-C. Filliâtre. A persistent union-find data structure. In *Proceedings of the 2007 workshop on Workshop on ML - ML '07*, page 37, Freiburg, Germany, 2007. ACM Press. ISBN 978-1-59593-676-9. doi: 10.1145/1292535.1292541. URL http://portal.acm.org/citation.cfm?doid=1292535.1292541.

[9] E. W. Dijkstra. *Selected writings on computing: a personal perspective*. Texts and monographs in computer science. Springer-Verlag, New York, 1982. ISBN 978-0-387-90652-2.

[10] G. G. Faust. Semiautomatic Translation of COBOL into HIBOL. Master's thesis, MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE, Feb. 1981. URL https://apps.dtic.mil/sti/citations/ADA099253.

[11] J. Jaffar. Efficient unification over infinite terms. *New Generation Computing*, 2(3):207–219, Sept. 1984. ISSN 0288-3635, 1882-7055. doi: 10.1007/BF03037057. URL http://link.springer.com/10.1007/BF03037057.

[12] A. Koster and R. A. Hatch. Prolog as Every Businessperson's Programming Language: A Survey of the Language. *Journal of Computer Information Systems*, 28(3):1–5, 1988. doi: 10.1080/08874417.1988.11646873. URL https://www.tandfonline.com/doi/abs/10.1080/08874417.1988.11646873.

[13] A. Krall. Implementation techniques for Prolog. In *WLP*, pages 1–15. Citeseer, 1994.

[14] H. P. Lovecraft, A. W. Derleth, and S. T. Joshi. *The Dunwich horror and others*. Arkham House, Sauk City, Wis, 1963. ISBN 978-0-87054-037-0.

[15] K. Marx and F. Engels. *Critique of the Gotha programme*. Cooperative Publishing Society of Foreign Workers in the U.S.S.R Moscow, 1937. Type: Book.

[16] B. Nystrom. Pratt parsers: Expression parsing made easy, 2011. URL http://journal.stuffwithstuff.com/2011/03/19/pratt-parsers-expression-parsing-made-easy/.

[17] J. E. Sammet. Basic elements of COBOL 61. *Communications of the ACM*, 5(5):237–253, May 1962. ISSN 0001-0782, 1557-7317. doi: 10.1145/367710.367721. URL https://dl.acm.org/doi/10.1145/367710.367721.

[18] K. Shirriff. Obama on sorting 1m integers: Bubble sort the wrong way to go, 2012. URL http://www.righto.com/2012/11/obama-on-sorting-1m-integers-bubble.html.

[19] O. I. Standard, V. OpenVMS, and U. Tru64. HP COBOL. 2005.

[20] D. H. Warren. An abstract Prolog instruction set. *Technical note 309*, 1983. URL https://ci.nii.ac.jp/naid/10007986292/en/. Publisher: SRI International.