Correct C# Grammar too Sharp for ISO

Vadim Zaytsev

Vrije Universiteit Amsterdam, The Netherlands,
vadim@cs.vu.nl

Introduction. The most used programming language nowadays is COBOL. At the Free Unversity in Amsterdam we have done numerous transformations on COBOL, parsed and transformed millions of lines of code. COBOL is standardised, but vendors usually deviate from the standard, making their own dialects. In order to parse code, we need a working grammar, which should be derived from the compiler documentation. However, documentation is never complete nor error-free, and special techniques are needed to obtain correct grammars: grammar recovery and grammar (re)engineering. One can argue whether this happens because of COBOL decades-long evolution and legacy.

Recently we started thinking about transforming C# code, too. C# is quite different from COBOL, it is a very sharp modern language, the latest big accomplishment in programming languages design. C# was produced by a big corporation and submitted as a specification to both ECMA International[1] and ISO[2]. C# compiler provided by Microsoft claims to fully implement the standard. Thus, one might think that this standard is of much better quality that COBOL's, making it easier to use it in parser construction. This research piece shows that it is not.

Specification quality: being sharp upon C#. The C# specification is almost 500 pages long, it is written in English, explains all language features in detail, and has an appendix with the formal language definition in a BNF-like form (the same formulae are used throughout the text). One might suppose it would be very easy to take that grammar and transform it into a working parser (which is needed for our re-engineering purposes). Unfortunately, it did not work out that easy: the C# specification's formal contents turned out to be unusable "as is". This means: no compiler. Actually, not even one line of code could have been parsed with that specification—so inconsistent was it.

In order to get to the parser, we took the BNF grammar apart from the text and put it into GDK[3], which is expected to generate SDF formulae from it (for use in the ASF+SDF Meta-Environment). This process showed that some BNF formulae are **informally** described ("separated by", "one of the following"), some are **redundant** (occur more than once, some sorts have identical definitions), some **incorrect** (e.g., forgotten "optional" marks), some **inconsistent** (formulae given in the text and in the appendix differ), some **non-intuitive** (e.g., expressions unintelligibly presented without priorities made implicit), some **idiosyncratic** (omnipresent YACCified constructions), some **ambiguous** (the

[1] *European Computer Manufacturers Association*. Here the ECMA-334 is meant.
[2] *International Organization for Standardization*. C# is ISO/IEC 23270:2003.
[3] *Grammar Deployment Kit* by C. Verhoef, R. Lämmel and J. Kort.

same code could be parsed in different ways), some just **non-lege artis**[4] (more compact grammar is better in many cases, same for parse trees).

Methods used and results obtained. GDK provides means for traceable grammar transformations. Around 1300 lines of GDK rewriting rules were needed to eliminate all irregularities, introduce absent definitions, disambiguate equivocal formulae (by reformulating them), deYACCify expressions, etc. The C# grammar was cut down from 281 sorts and 710 BNF formulae to about 172 sorts and 466 GDK rules (slightly extended EBNF). More than 100000 lines of C# code could been successfully parsed (SGLR) with it.

Grammar recovery—a technique for extracting a complete grammar out of an existing programming language's manual, a specification or a compiler's source code, assessing it, correcting, testing, an so on—has not been invented yesterday. This technique has been used in 1998 for the first time with PLEX[5], later it proved to be a success with COBOL—the language we mentioned in the first paragraph. There is ongoing work on Fortran and C grammars (which are considerably smaller than COBOL or C#). We make some grammars publicly available on the web: `http://www.cs.vu.nl/grammars/browsable/`.

The unofficial version of the C# standard, which can be found on the Microsoft website, is slighly less inconsistent—this fact shows the authors do care about such issues, but it does not give those corrections any official status.

Generalisation. This research has shown that grammar recovery techniques are needed with new languages as sharp as with the old ones. The problem is that most specifications are written by hand in a hope that the authors would be careful enough. Standards are very important part of the software industry nowadays, they should be reliable. It is possible to generate such specifications automatically from a complete working grammar and some textual remarks explaining it. This may take care of the consistency problem: all the BNF chunks in the standard will be correct, all the code examples will be valid, too. This has already been done with IBM VS COBOL II[6], and it can become a common practice, leading to tools like "Specification Deployment Kit" (non-existent yet).

It is well known that having a correct grammar of a programming language means that we can parse code, analyse it, re-engineer, compile, etc. Almost no transformational technique can be used without having a grammar. Besides that, a manual or an hyperlinked on-line documentation can be automatically generated from a complete well-commented grammar, too.

Conclusion. Language specification composed using "grammar hacking" is useless, but can be re-engineered into a neat grammar and a working parser. Conversely, language specification can be easily generated from a correct grammar. Grammar transformation techniques applied to this field lead to other advantages as well: backtracking, versioning, vendor specialising, subsetting, etc.

---

[4] *Lege artis*—Latin for "to be done in accordance with the rules of the art".

[5] *Programming Language for EXchanges*, a proprietary DSL for real time embedded software systems by Ericsson.

[6] Original *VS COBOL II Reference Summary* is here: `http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/IGYR1101/CCONTENTS`.