C STANDARD UNDEFINED BEHAVIOR VERSUS WITTGENSTEIN

VICTOR YODAIKEN

1. DEPRESSING AND FAINTLY TERRIFYING

Chris Lattner, the architect of the Clang/LLVM C compiler explained the effects of the C standard's "undefined behavior" (UB):

> *[...] UB is an inseperable part of C programming, [...] this is a depressing and faintly terrifying thing. The tooling built around the C family of languages helps make the situation less bad, but it is still pretty bad. The only solution is to move to new programming languages that dont inherit the problem of C. Im a fan of Swift, but there are others.*[4]

Also from Lattner[5]:

> *[...] many seemingly reasonable things in C actually have undefined behavior, and this is a common source of bugs in programs. Beyond that, any undefined behavior in C gives license to the implementation (the compiler and runtime) to produce code that formats your hard drive, does completely unexpected things, or worse*

Undefined behavior is an invention of the C standards committee to mark a large fragment of syntactically correct C code as outside the standard where compilers are free to apply any transformation — even changing the meaning of code elsewhere in the program. You might think that having the architect of a major compiler point out that the result of your standard is "depressing and faintly terrifying" and curable only by switching to another programming language would cause some consternation at the standards committee, but no. The WG14 C Standards organization is at work, expanding the realm of undefined behavior despite the warnings of compiler developers and the objections of the most prominent C developers and academics.

- Dennis Ritchie called an early version of UB "*a license for the compiler to undertake aggressive optimizations that are completely legal by the committee's rules, but make hash of apparently safe programs*"[7].
- One of the most important developers of cryptography code wrote: "*Pretty much every real-world C program is "undefined" according to the C "standard", and new compiler "optimizations" often produce new security holes in the resulting object code*" [1].
- In summary that leans over backwards to find excuses for UB, John Regehr wrote: *One suspects that the C standard body simply got used to throwing behaviors into the undefined bucket and got a little carried away. Actually, since the C99 standard lists 191 different kinds of undefined behavior, its fair to say they got a lot carried away.*[6].

Date: June 2018.
Key words and phrases. C, undefined behavior, compiler, wg14.

- And Linus Torvalds wrote: *The fact is, undefined compiler behavior is never a good idea. Not for serious projects.* [8].

2. Slow moving train wreck

Discussion of this issue or of practitioner objections in WG14 records is hard to find. What we do find is proposals to greatly expand the realm of undefined behavior[3]. And there is no doubt that the standards embrace of UB has created a slow motion train wreck.

- A security failure in the Linux kernel was generated by the compiler silently removing a check for a null pointer several lines below a UB reference to a null pointer.
- Code that checks for overflow in a signed variable can be silently "optimized" to remove the check because overflow of integer variables is UB.
- The C standard committee had to hack up its own standard with a special case when it realized it had made it impossible to write "memcpy" in C[2].
- Under pressure from, particularly Linux, the GCC compiler has introduced a number of flags to turn off UB behavior - essentially forking the language.

As Lattner wrote, C code that appears reasonable, that causes no compiler errors or warning and that conforms to wide practice can be arbitrarily changed once the compiler believes it has detected UB somewhere in the same neighborhood.

The rationale for UB is that this license opens up compiler optimizations that would be otherwise impossible. This claim is not supported by any actual studies. Even for the kinds of trivial micro-benchmarks that are often cited, it looks like absolutely nobody is doing any tests. For example, an unsigned and signed variable in a C "for loop" have no performance difference despite the advantage that the undefined nature of signed overflow supposedly provides the compiler.

Even though UB has been around since Ritchie first objected to it, it has grown in scope as the C standard has grown in scope and complexity. Here's a good example from C11:

> *(6.5 6) The effective type of an object for an access to its stored value is the declared type of the object, if any.) If a value is stored into an object having no declared type through an lvalue having a type that is not a character type, then the type of the lvalue becomes the effective type of the object for that access and for subsequent accesses that do not modify the stored value. If a value is copied into an object having no declared type using memcpy or memmove, or is copied as an array of character type, then the effective type of the modified object for that access and for subsequent accesses that do not modify the value is the effective type of the object from which the value is copied, if it has one. For all other accesses to an object having no declared type, the effective type of the object is simply the type of the lvalue used for the access.*
>
> *(6.5 7) An object shall have its stored value accessed only by an lvalue expression that has one of the following types: a type compatible with the effective type of the object, a qualified version of a type compatible with the effective type of the object, a type that is the signed or unsigned type corresponding to the effective type of the object, a type that is the signed or unsigned type corresponding to a qualified version of the effective type of the object, an aggregate or union type that includes one of*

> *the aforementioned types among its members (including, recursively, a member of a subaggregate or contained union), or a character type*

You have to read the footnote to paragraph (6.5. 7) to have any idea at all what is being "specified".

> " *The intent of this list is to specify those circumstances in which an object may or may not be aliased.*"

Now consider this rule in light of the proposal of Dean Keaton that anything not precisely defined in the standard qualifies as UB - think about the galaxy of imprecise specification around these two paragraphs of murk.

No wonder that Linus Torvalds was not impressed.

```
The idiotic C alias rules aren't even worth discussing.
They were a mistake.  The kernel doesn't use some "C
dialect pretty far from standard C".  Yeah, let's just
say that the original C designers were better at their
job than a gaggle of standards people who were making
bad crap up to make some Fortran-style programs go faster.
  They don't speed up normal code either, they just introduce
undefined behavior in a lot of code.
  And deleting NULL pointer checks because somebody made
a mistake, and then turning that small mistake into a
real and exploitable security hole?  Not so smart either.
[8]
```

The situation is only going to get worse as link time optimization becomes common in compilers - extending the ability of the compilers to find UB across separate compilation borders.

3. FRUGALITY BE DAMNED

There's a unfortunate dynamic in the WG14 process in which the language definition expands in scope while shrinking in clarity. This dynamic comes from a lack of appreciation for the frugal semantics of the C language. In software, as in much other forms of engineering, only the most brilliant or lucky designers manage to leave the right things out. In particular, C and UNIX were so successful as much because of what was left out as because of what was put in. The injunction of Wittgenstein[9] is almost a cliche in programming design, but it is on point

> *What can be said at all can be said clearly; and whereof one cannot speak thereof one must be silent.*

The approach of the WG14 committee has appears to have been based on the misconception that this economy in specification was a problem to be solved by expanding the scope of things covered by the standard. The alias rules above are a good example. C had no aliasing rules because it gained an advantage by permitting programmers to use pointers flexibly. Lack of aliasing rules could have been argued as too costly in terms of both programmer errors and forsaken compiler optimizations, but that could have been addressed by refining the restrict keyword or providing some other opt-in semantics instead of trying to build a more complex notion of type into C. As another example, C did not define anything about concurrent tasks and parallelism, even though the designers of C intended to use it with interrupt handlers and processes and multi-processors, and

did so with great success. By omitting the semantics of concurrent and parallel computation, the C design licensed programmers to develop many different solutions outside the framework of the language. Interrupt handlers, POSIX processes and threads, coroutines, SMP, the event models of current generation web servers, GPUs and vector processing: the minimalist design of the language meant that it did not get in the way of system designers and programmers. But the WG14 committee is now attempting to grow the language to cover the semantics of threaded code. Necessarily, this specification will have imprecision, opening the way to even more UB and it looks like it will tie the language to an awkward model of synchronization (via mutexes) and import into the language information about processor design that may well be obsolete in a few years.

Either C will fade away as Lattner and many others hope, the standard will change, or there will be a successful fork to produce a more precise and flexible standard.

REFERENCES

[1] D.J. Bernstein. *boringcc*. https://groups.google.com/forum/m/\#!msg/boring-crypto/48qa1kWignU/o8GGp2K1DAAJ.

[2] Derek Jones. *How indeterminate is an indeterminate value*. http://shape-of-code.coding-guidelines.com/2017/06/18/how-indeterminate-is-an-indeterminate-value/.

[3] David Keaton. *Implicit Undefined Behavior in C*. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2248.pdf.

[4] Chris Lattner. *GEP with a null pointer base*. http://lists.llvm.org/pipermail/llvm-dev/2017-July/115149.html. 2016.

[5] Chris Lattner. *What every C programmers should know about undefeind behavor 1/3*. http://blog.llvm.org/2011/05/what-every-c-programmer-should-know.html.

[6] John Regehr. *A Guide to Undefined Behavior in C and C++, Part 1*. https://blog.regehr.org/archives/213.

[7] Dennis Ritchie. *noalias comments to X3J11*. http://www.lysator.liu.se/c/dmr-on-noalias.html.

[8] Linus Torvalds. *Re: [isocpp-parallel] Proposal for a new memory order*. https://gcc.gnu.org/ml/gcc/2016-02/msg00381.html.

[9] L. Wittgenstein and C.K. Ogden. *Tractatus Logico-philosophicus*. International library of psychology, philosophy, and scientific method. Routledge, 1990. ISBN: 9780415051866. URL: https://books.google.com/books?id=TE0nC-MfRz4C.

AUSTIN TEXAS.
E-mail address: victor.yodaiken@gmail.com