n2278 - Proposal to clarify undefined behavior range for implementations

.

Rationale:

This Standard's treatment of "undefined behavior", like the rest of the Standard, relies on an expectation that implementations are cooperating with the programmer and respecting the C abstract semantics. For example, overflow is undefined behavior, yet 5.1.2.3 Example 6, the Standard limits reordering certain expressions when the implementation produces traps on arithmetic overflows. The purpose of "undefined behavior" is to relieve the translator from the burden of hiding implementation differences and from making safety checks that are the responsibility of the C programmer. Unfortunately, some implementations are interpreting undefined behavior to license arbitrary transformations that violate the abstract semantics.

As an example, some implementations will roll-over on arithmetic overflow but, at the same time, delete programmer checks for roll-over as an "optimization" because overflow is "undefined behavior". The wording of the Standard does not support this interpretation: "*possible undefined behavior ranges from"*

- *ignoring the situation completely with unpredictable results*" - which case the underlying processor architecture might increment in modular arithmetic or trap (which may or may not be handled) or do something else, but the translation "ignore[s] the situation" and produces code without making use of knowledge of any undefined behavior;
- "*to behaving during translation in a documented manner characteristic of the environment*";
- " *to terminating a translation or execution (with the issuance of a diagnostic message)*".

This range is very wide and provides implementations with latitude for optimization but
it does not support conforming implementations that do not ignore the situation but neither behave in a documented manner characteristic of the environment nor terminate the translation and instead produce essentially arbitrary or inconsistent translation..

Arbitrary code translation justified by undefined behavior makes C programming a hazardous endeavor where the implementation is full of unpleasant surprises. Additionally, the more radical interpretations of undefined behavior are causing a fork in the language. Currently, the Linux Operating System requires its translators to respect flags to turn off a number of "optimizations" that are at least partially based on detecting undefined behavior. The existence of these flags and their wide use indicates both that a great deal of existing practice depends on disabling undefined behavior based "optimization" and that accommodating a more modest use of UB directed translation is acceptable to developers of implementations.

It has been argued that permitting implementations to violate the constraints of the abstract machine in the presence of undefined behavior is necessary for high levels of optimization, but there are no published studies validating that claim, and, indeed, the Linux kernel code is highly optimized despite its extensive use of the translator options described in the preceding paragraph. Limiting UB directed "optimizations" will incentivize implementations to optimize within the C semantics and make C more reliable without requiring the major rewrite of the Standard suggested in https://blog.regehr.org/archives/1180 (which motivated this proposal).

Proposed change.

3.4.3

undefined behavior
1 behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no requirements. Note that the use of a nonportable or erroneous program construct or use of erroneous data should not affect the translation or implementation of conforming constructs in the program or the uses of acceptable data.

2 NOTE Possible undefined behavior ranges from ignoring the situation completely with unpredictable results, to behaving during translation or program execution in a documented manner characteristic of the environment (with or without the issuance of a diagnostic message), to terminating a translation or execution (with the issuance of a diagnostic message). This range is exhaustive, not inclusive. Conforming implementations cannot violate semantics of the C abstract machine in conforming program constructs using acceptable data due to the presence of undefined behavior somewhere else in the program. Use of a nonportable or erroneous program construct may produce undefined behavior but should not affect translation of other program constructs. By 5.1.2.3.4 “An actual implementation need not evaluate part of an expression if it can deduce that its value is not used “ but merely deducing that undefined behavior follows from use of a value does not allow the implementation to assume that the value is not used or to incorrectly assume anything else about program operation.

3 EXAMPLE An example of undefined behavior is the behavior on integer overflow.

4 EXAMPLE Program constructs that check to see if an integer value has overflowed or an array index is past the limit of the array cannot be deleted or otherwise ignored during translation unless the overflow or out of bounds index is actually impossible in the implementation. 5.1.2.3 applies to conforming program constructs even if other program constructs produce undefined behavior.

5 EXAMPLE If the value of the right operand of a shift is negative, the result is undefined but a conforming implementation cannot deduce that the value of a conforming expression is not negative merely from its use as the right operand of a shift instruction.

6 EXAMPLE Code that checks to see if a pointer value is null cannot be omitted during translation solely on the basis that the pointer is dereferenced before the check and that earlier dereference would produce undefined behavior if the pointer value was null.

7 NOTE Implementation of signed integer addition in an environment that generated overflow traps on overflow follows the specification as would implementations where the addition simply rolled over the integer value. However injecting for example, an infinite loop that would be software triggered on overflow would not fit into the range of possible undefined behaviors because it would not be “characteristic of the environment” even if documented (as required).