Quantize - to Underflow or Not

Clarifying Inexact, Tiny, and Underflow

Eric Schwarz

How many **pints** of beer left?

1/8 drop is left

Does anyone want me to describe it to
16 or 34 significant decimal digits?

Or just buy another round?

How many **drops** of beer left?

1/8 drop is left

Should I underflow?

I think someone asked me in the wrong units?
(scale could have been passed as an argument
into the beer_left subroutine)

Quantize(x,y) to integer (e=0) similar to Round to Integral Value

- x = Dmin = 1 x 10**Xmin
- y = 10**0

- Intermediate result after align =
- 0000… 000 . G=0, S=1 x 10**0

- No underflow detected
- Rounds to 0 x 10**0 or 1 x 10**0

Quantize(x,y) to subnorm

- x = Dmin = 1 x 10**Xmin
- y = 10**Xmin+2

- Intermediate result after align =
- 0000… 000 . G=0, S=1 x 10**Xmin+2

- underflow detected?
- Rounds to
  - 0 x 10**Xmin+2 or (zero, inexact)
  - 1 x 10**Xmin+2 (tiny, inexact)

How is underflow detected?

- **What is the intermediate result?**
  1. Important if underflow determined prior to rounding
  2. Shouldn't depend on rounding mode
  3. Round to Integral Value should not Underflow
  4. Should rounding be considered part of operation (see next foil)

  A 0 x 10**Xmin+2 (no underflow, exact zero)
  B 1 x 10**Xmin (underflow even for e=0, round integral value)
  C x 10**Xmin+2 (weird, but may be best definition)

- Nmin is when q + Significant Digits = Xmin+p

What if Rounding part of Operation?

- x = Dmin = 1 x 10**Xmin y = 10**Xmin+2
- Operation result
  - 0 x 10**Xmin+2 or (zero) no underflow
  - 1 x 10**Xmin+2 (tiny) underflow (trap enabled)
    - Is inexact flag definition different than loss of accuracy for underflow? Inexact flag set by input != output, LOA from sect 7.4 never occurs due to result is same for different precisions.
- Pros
  - Consistent with Round to Integral Value
  - No Wrap of Zero Value
  - Underflow dependent on value rather than representation (SD)
- Cons
  - Underflow / tiny detection depends on rounding mode so appears inconsistent with detection prior to rounding
    - Depends on what constitutes the operation

Backup Details

from Ron Smith

- The current IEEE definition implies **three** different types of rounding and **two** types of inexact. This is causing a good bit of confusion, especially in the definition of underflow.

- **Operation Rounding** (previously used only by Round Floating-Point Number to Integer Value, section 5.5). For this type of rounding, the argument is rounded to an integer before applying underflow tests.

- **Normal Rounding** computes a result with unbounded exponent range but bounded precision.

- **Denormalization Rounding** (only performed when underflow trap is disabled) computes a result with bounded exponent range and bounded precision.

- Note that the use of terms "before rounding" and "after rounding" in the tininess tests for underflow are a bit misleading as the implied sequence of execution is as follows:

- 1. **Operation Rounding**
- 2. Tininess Test Before Rounding
- 3. **Normal Rounding**
- 4 Tininess Test After Rounding
- 5. **Denormalization Rounding**

- Two definitions for Inexact:

- **Inexact Definition A**: Signaled when rounding changes the value.

- **Inexact Definition B**: Exists when the delivered result differs from what would have been computed were both exponent range and precision unbounded.

- Example of **Inexact Definition A**:
- Section "4. Rounding ....Rounding takes a number regarded as infinitely precise and, if necessary, modifies it to fit in the destination's format while signaling the inexact exception (7.5)."

- (The last phrase strongly implies that modification and signaling inexact go together.)

- Example of **Inexact Definition B**:
- Section "7.4. Underflow ...2. An inexact result - when the delivered result differs from what would have been computed were both exponent range and precision unbounded. (This is the condition called inexact in 7.5).

- (It could be claimed that the last sentence is the official definition of inexact. But see section 7.5 below.)
- Section "7.5. Inexact ... If the rounded result of an operation is not exact ..."
- This is a circular definition, so it could be interpreted either way.

- (If this were a democracy - we have one for A, one for B, and one undecided.)
- The two definitions of inexact represent a very fundamental problem and the standard needs to clarify which direction it is heading in this area.
- The principle of **Inexact Definition A** was used to in 854-1987 to state: "5.5 Round Floating-Point Number to Integral Value. ... This operation signals inexact whenever its argument differs in value from its result."
- The principle of **Inexact Definition B** was used in DRAFT IEEE Standard for Floating-Point Arithmetic 2003 August 12 10:20 to state: "5.5. Round Floating-Point Number to Integer Value .... This operation never generates an inexact exception."