The conversion of a binary format floating point number to an integer or decimal representation that is too big for the format in question is an invalid operation, but it cannot deliver a NaN since there is no floating point destination for the result.

Exercise 7.8 *Extend Exercise* 4.3 *to the case where either x or y may be ± 0, $\pm\infty$, or* NaN*, and the result may be "unordered".*

Overflow

Traditionally, *overflow* is said to occur when the exact result of a floating point operation is finite but with an absolute value that is larger than the largest floating point number. As with division by zero, in the days before IEEE arithmetic was available the usual treatment of overflow was to set the result to (plus or minus) the largest floating point number or to interrupt or terminate the program. In IEEE arithmetic, the standard response to overflow is to deliver the correctly rounded result, either $\pm N_{\max}$ or $\pm\infty$. The range of numbers that round to $\pm\infty$ depends on the rounding mode; see Chapter 5.

To be precise, overflow is said to occur in IEEE arithmetic when the exact result of an operation is finite but so big that its correctly rounded value is different from what it would be if the exponent upper limit $E_{\max}$ were sufficiently large. In the case of *round to nearest*, this is the same as saying that overflow occurs when an exact finite result is rounded to $\pm\infty$, but it is not the same for the other rounding modes. For example, in the case of *round down* or *round towards zero*, if an exact finite result x is more than $N_{\max}$, it is rounded down to $N_{\max}$ no matter how large x is, but overflow is said to occur only if $x \geq N_{\max} + \mathrm{ulp}(N_{\max})$, since otherwise the rounded value would be the same even if the exponent range were increased.

Gradual Underflow

Traditionally, *underflow* is said to occur when the exact result of an operation is nonzero but with an absolute value that is smaller than the smallest normalized floating point number. In the days before IEEE arithmetic, the response to underflow was typically, though not always, *flush to zero*: return the result 0. In IEEE arithmetic, the standard response to underflow is to return the correctly rounded value, which may be a subnormal number, ± 0 or $\pm N_{\min}$. This is known as *gradual underflow*. Gradual underflow was and still is the most controversial part of the IEEE standard. Its proponents argued (and still do) that its use provides many valuable arithmetic rounding properties and significantly adds to the reliability of floating point software. Its opponents argued (and still do) that arithmetic with subnormal numbers is too complicated to justify inclusion as a hardware operation which will be needed only occasionally. The ensuing debate accounted for much of the delay in the adoption of the IEEE standard. Even today, some IEEE compliant microprocessors support gradual underflow only in software. The standard gives several options for defining exactly when the underflow exception is said to occur; see [CKVV02] for details.

The motivation for gradual underflow can be summarized very simply: compare Figure 3.1 with Figure 4.1 to see how the use of subnormal numbers fills in the relatively large gap between $\pm N_{\min}$ and zero. The immediate consequence is that the worst case absolute rounding error for numbers that underflow to subnormal numbers is the *same* as the worst case absolute rounding error for numbers that round to $N_{\min}$. This is an obviously appealing property.

Consider the following subtraction operation, using the IEEE single format. The