Operator precedence cheat sheet for Isabelle/HOL (quick and dirty, little quality control, version 2012-Nov-12). Follow-up of the discussion at the Isabelle mailing list in November 2012, `https://lists.cam.ac.uk/pipermail/cl-isabelle-users/2012-November/msg00020.html`. Entries without a fixed precedence are still rough guesstimates (that may need further correction).

Class	Precedence	Symbols and the minimum precedence of symbols that may be contained in non-terminals
function		application of user-defined function, field dereference (record)
function	900	(900 ⦇ 0 ⦈ (record update))
function	100	(100 nth (list) 101)
type		"set" type constructor, "list" type constructor
function	70	(70 ∩ (set) 71), (70 * (nat) 71)
function	65	(65 ∪ (set) 66), (65 + (nat) 66), (65 − (nat) 66), (66 # (list) 65), (66 @ (list) 65)
relation	50	(50 = 51), (50 ∈ 51), (51 < 51), (51 ≤ 51)
logic	40	(¬ 40)
logic	35	(36 ∧ 35)
logic	30	(31 ∨ 30)
logic	25	(26 ⟶ 25), (26 ↔ 25)
logic	10	(∀ 0 . 10), (∃ 0 . 10), (if 0 then 0 else 10), (case 0 of 10), (let 0 in 10)
type	3	(4 :: 0)
logic	3	(λ 0 . 3)
metalogic	2	(3 ≡ 2)
metalogic	1	(2 ⟹ 1)
type	0	(1 ⇒ 0)
metalogic	0	(⋀ 0 . 0)

Table 1: Higher symbols have higher precedence. Precedences given where known to me, by use of the Isabelle command "print_syntax" - you can issue this command for example anywhere in an empty theory or an existing theory. "print_syntax" also shows precedence for a lot of symbols not given in the table for brevity (say, exponentiation of natural numbers). Following the convention of the output of "print_syntax", associativity of binary operators is determined by that a non-terminal in the right-hand side production rule of a grammar only may be expanded by non-terminals with the same or higher precedence. Hence, for example addition on the natural numbers is left-associative, as its production still allows expansion to the left of the "+" operator. Conversely, for example list concatenation is right-associative. "<" is not associative at all (no expansion allowed either way). See also the post by Holger Gast for a more detailed explanation (`https://lists.cam.ac.uk/pipermail/cl-isabelle-users/2012-November/msg00032.html`). Classes are my own interpretation.

Note on classes:

function if it is a function (type constructors "set", "list" could also be seen as functions)

relation if it is a relation

logic if it is object logic

metalogic if it is meta logic