vararg function; they are explained in §3.4.10.

Binary operators comprise arithmetic operators (see §3.4.1), relational operators (see §3.4.3), logical operators (see §3.4.4), and the concatenation operator (see §3.4.5). Unary operators comprise the unary minus (see §3.4.1), the unary **not** (see §3.4.4), and the unary *length operator* (see §3.4.6).

Both function calls and vararg expressions can result in multiple values. If a function call is used as a statement (see §3.3.6), then its return list is adjusted to zero elements, thus discarding all returned values. If an expression is used as the last (or the only) element of a list of expressions, then no adjustment is made (unless the expression is enclosed in parentheses). In all other contexts, Lua adjusts the result list to one element, discarding all values except the first one.

Here are some examples:

```
f()                 -- adjusted to 0 results
g(f(), x)           -- f() is adjusted to 1 result
g(x, f())           -- g gets x plus all results from f()
a,b,c = f(), x      -- f() is adjusted to 1 result (c gets nil)
a,b = ...           -- a gets the first vararg parameter, b gets
                    -- the second (both a and b can get nil if there
                    -- is no corresponding vararg parameter)

a,b,c = x, f()      -- f() is adjusted to 2 results
a,b,c = f()         -- f() is adjusted to 3 results
return f()          -- returns all results from f()
return ...          -- returns all received vararg parameters
return x,y,f()      -- returns x, y, and all results from f()
{f()}               -- creates a list with all results from f()
{...}               -- creates a list with all vararg parameters
{f(), nil}          -- f() is adjusted to 1 result
```

Any expression enclosed in parentheses always results in only one value. Thus, `(f(x,y,z))` is always a single value, even if `f` returns several values. (The value of `(f(x,y,z))` is the first value returned by `f` or **nil** if `f` does not return any values.)

3.4.1 – Arithmetic Operators

Lua supports the usual arithmetic operators: the binary `+` (addition), `-` (subtraction), `*` (multiplication), `/` (division), `%` (modulo), and `^` (exponentiation); and unary `-` (mathematical negation). If the operands are numbers, or strings that can be converted to numbers (see §3.4.2), then all operations have the usual meaning. Exponentiation works for any exponent. For instance, `x^(-0.5)` computes the inverse of the square root of `x`. Modulo is defined as

```
a % b == a - math.floor(a/b)*b
```

That is, it is the remainder of a division that rounds the quotient towards minus infinity.

3.4.2 – Coercion

Lua provides automatic conversion between string and number values at run time. Any arithmetic operation applied to a string tries to convert this string to a number, following the rules of the Lua lexer. (The string may have leading and trailing spaces and a sign.) Conversely, whenever a number is used where a string is expected, the number is converted to a string, in a reasonable format. For complete control over how numbers are converted to strings, use the `format` function from the string library (see `string.format`).

3.4.3 – Relational Operators