Regardless of the approach, these `t` × `f` elements will have to be accommodated. The architectural problem is how we group them into modules to facilitate extension and reuse. This issue is not discussed in the article and presentation. Of course, the matter is not critical for small `t` and `f`; then all the definitions can be packed into a single module. This takes care of extendibility in a simple way:

- To add a basic combinator `c`, add `f` definitions of the above form, one for each existing operation.
- To add an operation `O`, add `t` definitions, one for each existing combinator.

This approach does not scale well; for larger developments, it will be necessary to divide the system into modules; the extendibility problem then becomes how to make sure that such modifications affect as few modules as possible.

Even with fairly small `t` and `f`, the one-module solution does not support reusability: if another program only needs a subset of the operations and combinators, it would suffer the usual dilemma of primitive modularization techniques:

Charybdis
: Copy-paste the relevant parts, but then risk forgetting to update the derived modules when something changes in the original (possibly for such a prosaic reason as a bug fix).

Scylla
: Use a module inclusion facility, as provided by many languages, to make the contents of an existing module available to a new one; but you end up loaded with a bigger baggage than necessary, which complicates updates and may cause conflicts (assuming the derived module defines a new combinator or function and a later version of the original module introduces a clashing definition).

These observations remind us in passing that reusability is closely connected to extendibility. An online critique of the OCaml functional language (Steingold 2007) takes a concrete example:†

> You cannot easily modify the behavior of a module outside of it. Suppose you use a Time module defining `Time.date_of_string`, which parses ISO8601 basic format ("YYYYMMDD"), but want to recognize ISO8601 extended format ("YYYY-MM-DD"). Tough luck: you have to get the module maintainer to edit the original function—you cannot redefine the function yourself in your module.

As software grows and changes, another aspect of reuse becomes critical: reuse of common properties. Along with European options, the article introduces "American options." Described as combinators, they have different signatures (`Date → Contract → Contract` and `(Date, Date) → Contract → Contract`). One suspects, however, that the two kinds of option have a number of

† This citation is slightly abridged. Inclusion of the citation does not imply endorsement of other criticism on that page.