Examples of *combinators* on contracts include: `or`, such that acquiring the contract (`or c1 c2`) means acquiring either of `c1` and `c2`, and expiring when both have expired; `anytime`, such that (`anytime c`) can be acquired at any time before the expiration of `c`, and expiring whenever `c` expires; `truncate`, such that (`truncate t c`) is like `c` except that it expires at the earlier of `t` and the expiry of `t`; and `get`, so that acquiring (`get c`) means acquiring `c` at its expiry date. The paper lists about a dozen such basic combinators on contracts, and others on observables and dates. They make it possible to define advanced financial instruments such as a "European option" in a simple way:

```
european t u   = get (truncate t (or u zero))
```

*Operations* include the expiry date of a contract and—the most important practical benefit expected from all this modeling effort—its value process, a time-indexed sequence of expected values. As with the sugar content of a pudding, the functions are defined by case analysis on the basic constructors. Here are the cases involving the preceding basic elements and combinators for the operation `H`, which denotes the expiry date or "horizon":

```
H (zero)         =  ∞   -- Where ∞ is a special value with the
        expected properties
H (or c1 c2)     =  max (H (c1), H (c2))
H (anytime c)    =  H (c)
H (truncate t c) =  min (t, H (c))
H (get c)        =  H (c)
```

The rules yielding value processes follow a similar structure, although the righthand sides are more sophisticated, involving financial and numerical computations. For more examples of applying combinators and functional programming ideas to financial applications, see Frankau (2008).

## Assessing the Modularity of Functional Solutions

The preceding presentation, while leaving aside many contributions of the presentation and especially the article, suffices as a basis for discussing architectural features of the functional approach and comparing them with the OO view. We will freely alternate between the pudding example (which makes the ideas immediately understandable) and financial contracts (representative of real applications).

### Extendibility Criteria

As pointed out by the presentation, the immediate architectural benefit is that it is easy to add a new combinator: "When we define a new recipe, we can calculate its sugar content with no further work." This property, however, is hardly a consequence of using a functional programming approach. The insight was to introduce the notion of a combinator, which creates pudding and pudding parts—or contracts—from components that can either be atomic or themselves result from applying combinators to more elementary components.