> **NOTE**
>
> **We share the reader's alarm at the unappetizing nature of the example, especially coming from a Paris-based author. The sympathetic explanation is that the presentation was directed to a foreign audience of which it assumed, along with unfamiliarity with the metric system, barbaric culinary habits. The present discussion relies on the assumption that bad taste in desserts is not a sufficient predictor of bad taste in language and architecture paradigms.**

The nonleaf nodes of the tree represent combinators, applied to the subtrees. For example, "Take" is a combinator that assumes two arguments, a pudding part ("Cream" on the left, "Oranges" on the right) and a quantity ("1 pint" and "6"); the result of the application, represented by the tree node, is a pudding part or a pudding made of the given quantity of the given part.

It is also possible to write out such a structure textually, using a mini-"domain-specific language" (DSL) "for describing puddings" (boldface is added for operators):

```
"salad      = on_top_of topping main_part"      -- Changed from
        "OnTopOf" for consistency
"topping    = whipped (take pint cream)
main_part   = mixture apple_part orange_part
apple_part  = chopped (take 3 apples)
orange_part = optional (take 6 oranges)"
```

This uses an anonymous but typical—the proper term might be "vanilla"—variant of functional programming notation, where function application is simply written as function args (for example, `plus a b` for the application of `plus` to `a` and `b`) and parentheses serve only for grouping.

With this basis, it becomes a piece of cake to define an operation such as sugar content (`S`) by case analysis on the combinators (similar to defining a mathematical function on recursively defined objects by using a definition that follows the same recursive structure):

```
"S  (on_top_of p1 p2)   = S (p1) + S (p2)
S   (whipped p)         = S (p)
S   (take q i)          = q * S(i)
etc."
```

Not clear (to us) from the "etc." is how operators such as `S` deal with the `optional` combinator; there has to be some way of specifying whether a particular concoction has the optional part or not. This issue aside, the approach brings the benefits that the presentation claims for it:

- "When we define a new recipe, we can calculate its sugar content with no further work.
- Only if we add new combinators or new ingredients would we need to enhance `S`."

The real goal, of course, is not pudding but contracts. Here the presentation contains a sketch of the approach but the article is more detailed. It relies on the same ideas, applied to a more interesting set of elements, combinators, and operations.

The *elements* are financial contracts, dates, and observables (such as a certain exchange rate on a certain date). Examples of basic contracts include `zero` (can be acquired at any time, no rights, no obligations) and `one (c)` for a currency `c` (immediately pays the holder one unit of `c`).