ИСПОЛЬЗОВАНИЕ ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ В АЛГОРИТМИЗАЦИИ

Е.П. Ельников, Е.А. Житковский, И.А. Тонко
Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь
Ролич О.Ч. – канд.техн.наук, доцент

Интенсиональность. Сущности в программировании выступают в трех ролях: сущности, средства построения сущностей, средства применения средств построения сущностей. Эта не имманентная характеристика сущности, а внешняя. Определяется не сущностью самой по себе, а способом её использования в конкретном контексте. Сущности в программировании - интенсиональны. Пример - функция: чаще всего выступает средством построения объектов данных, но в может и сама выступать объектом построения.

Композиционность. Композиции строят более сложные программные сущности из относительно простых. Чаще всего имеют в виду композиции программ, но не менее важны композиции данных, данных с программами и т.д. Обычно их относят к интенсиональному типу средств построения. Но не менее важно рассматривать их и как средства применения и, в частности, управления применением: циклическая композиция управляет выполнением оператора-тела, который выступает средством построения объектов данных, объект, реализующий идиому RAII, управляет применением заданного действия. Всякое ответвление программирования характеризуется своим набором композиций. Эти композиции выступают инструментами сочленения более сложных единиц из простых и управляют применением частей в составе целого в процессе его работы. Именно композиции определяют суть и дух всякого программирования.

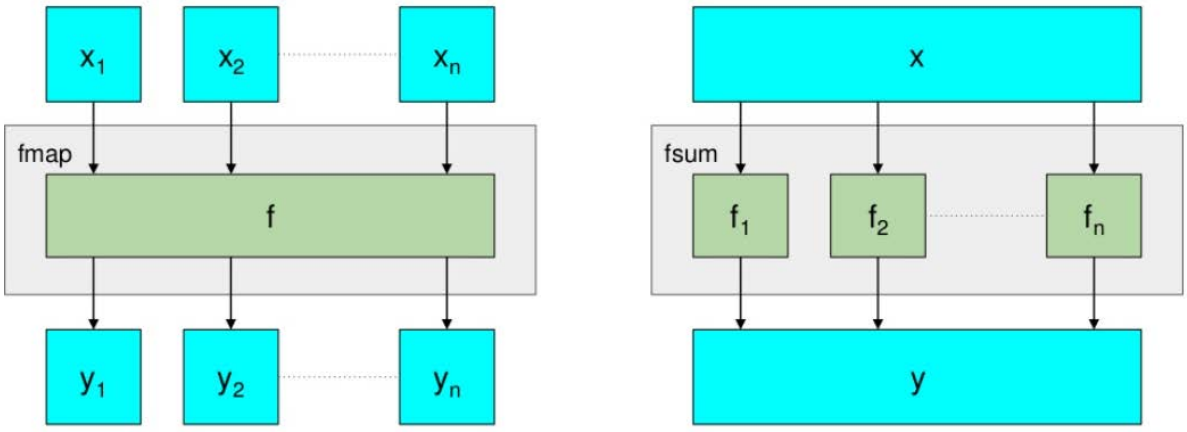

Рисунок 1 – Пример использования суммы функций

Пример. Пусть дана совокупность функций каждая из которых можеть дать искомый результат или неудачу. Тогда процесс обработки может выглядеть следующим образом:

- Попытаться извлечь параметр hostname из файла настроек в текущей директории;
- Попытаться извлечь hostname из настроек в домашней директории пользователя;
- Попытаться извлечь hostname из глобального файла настроек;
- Взять hostname по умолчанию, прошитый в программе.

Каждая операция, кроме последней, может окончиться неудачей: отсутствует файл или в файле нет такого параметра. Таким образом, чтобы получить правильный hostname, нужно пытаться применять эти функции в том же порядке одну за другой до первой удачи.

Определение суммы функций. Пусть дана совокупность функций $f_1,\ldots, f_n$ типа $A_1,\ldots, A_m \to T^?$. Их сумма $h = \sum[f_1,\ldots, f_n] = f_1 \oplus \ldots \oplus f_n$ есть функция того же типа. Если $f_k(x) = \bot$ для всех $k \in \{1, \ldots, n\}$, то $h(x) = \bot$. Иначе $h(x) = f_k(x)$, где $k \in \{1, \ldots, n\}$ - наименьшее такое число, что $f_k(x) \neq \bot$, $f_i(x) = \bot$ для всех $i \in \{1, \ldots, k-1\}$. Неформально говоря: сумма функций по очереди пробует применять все функции-слагаемые до первой удачи. Сумма функций терпит неудачу, если неудачу терпят все слагаемые.

Свойства суммы функций:

- Некоммутативность: $(p[a, y] \oplus p[a, z])(a) = y$, но $(p[a, z] \oplus p[a, y])(a) = z$.
- Коммутативность при условии: если $f \approx g$, то $f \oplus g = g \oplus f$.
- Ассоциативность: $(f \oplus g) \oplus h = f \oplus (g \oplus h)$.
- Левый нуль: $\bot \oplus f = f$.
- Правый нуль: $f \oplus \bot = f$.
- Идемпотентность: $f \oplus g \oplus g = f \oplus g$ (в программировании работает только для чистых функций).
- Неподвижность максимума слева: если $t \in Tot$, то $t \oplus f = t$.
- Сохранение максимума справа: если $t \in Tot$, то $(f \oplus t) \in Tot$.
- Монотонность: $f \preccurlyeq f \oplus g$.
- Поглощение меньшего большим: если $f \preccurlyeq g$, то $f \oplus g = g$.

Для реализации алгоритма суммы функций необходимо сформулировать ее рекуррентное определение. База (сумма из 0 слагаемых есть 0): $\sum[\,] = \bot$. Шаг: выразить сумму n слагаемого через сумму n-1 слагаемых ($n > 0$): $h = \sum [f_1,\ldots, f_n]$ - это такая функция, что для любого x: если $f_1(x) = y \neq \bot$, то $h(x) = y$. В противном случае $h(x) = g(x)$, где $g = \sum [f_2,\ldots, f_n]$. Пользуясь данным определением можно реализовать сумму функций в рамках любого языка программирования, что позволит пользоваться преимуществами функционального программирования в объектно-ориентированном языке, например, в C++.