1. Модель функционально-потоковых параллельных вычислений

Модель вычислений (МВ), определяющая динамические свойства ФЯПП, имеет следующие основные характеристики [KKL, ККЛ-1997, ЛКК-1996, ЛККВ]:

- В основе модели лежит управление вычислениями по готовности данных. Сами вычисления протекают внутри бесконечных ресурсов, что позволяет неявно задавать параллелизм без анализа ресурсных конфликтов;
- Выбор операций и аксиом, определяющих базовый набор функций, ориентирован на наглядное текстовое представление информационного графа программы при его описании на языке программирования.

Первое требование обеспечивает ресурсную независимость предлагаемой модели, что позволяет описать максимальный параллелизм, ограниченный только информационными связями, присущими данной задаче. Это сводит перенос написанной программы на любую вычислительную систему к распределению ресурсов в соответствии с архитектурой. Подобный подход используется также в ряде известных схем потока данных (СПД) [Алгоритмы], ориентированных на рекурсивное описание программ, обрабатывающих только один входной поток данных и не поддерживающих их конвейерное продвижение. В связи с отсутствием циклических конструкций граф данной модели является ациклическим.

Использование текстового представления программ связано с трудностями описания информационного графа, что привело к синтаксису языка, несколько отличающемуся от общепринятого. Кроме того, в языке отсутствуют вентили, обеспечивающие условную передачу данных в традиционных СПД [Денис, Arvind]. Эти вентили трудно структурировать при текстовом описании программ без использования дополнительной синхронизации информационных потоков.

1.1. Общие принципы организации модели

Модель задается тройкой

$$\mathbf{M = (G, P, S_0),}$$

где **G** - ациклический ориентированный граф, определяющий информационную структуру программы (ее информационный граф), **P** - набор правил, определяющих динамику функционирования модели (механизм формирования разметки), $\mathbf{S_0}$ - начальная разметка.

Информационный граф

$$\mathbf{G = (V, A),}$$

где **V** - множество вершин определяющих программо-формирующие операторы, а **A** - множество дуг, задающих пути передачи информации между ними.

Вершины графа, соответствующие программо-формирующим операторам, обеспечивают информационные преобразования данных, их структуризацию и размножение. Существуют следующие типы операторов:

- оператор интерпретации;
- константный оператор;
- оператор копирования данных;
- оператор группировки в список;
- оператор группировки в параллельный список;
- оператор группировки в список задержанных вычислений.

Операторы используются для построения информационного графа, определяющего функцию (процедуру), являющуюся независимым программным модулем. Этот граф имеет множество входных информационных дуг $\mathbf{A_0}$, на которые поступают аргументы, задающие исходные данные, необходимые для организации вычислений. Поступление этих аргументов фиксируется в начальной разметке графа. Существует также один выход, определяющий

результат. Граф с несколькими входными дугами и одной выходной дугой назовем допустимым графом. Вычисления заканчиваются, когда все дуги допустимого графа, включая и выходную дугу, окажутся размеченными.

Динамика выполнения операторов задается механизмом продвижения начальной разметки графа по дугам модели. Разметка дуги задается вектором:

$$M_i = (N, R),$$

где **N** - кратность дуги, определяющая количество независимых наборов данных, полученных в результате выполнения оператора, выход которого соединен с этой дугой; **R** - вектор данных $(r_1, r_2, \ldots r_N)$, полученный в ходе вычислений.

Наличие разметок на всех дугах некоторой вершины графа позволяет запустить ее и получить выходную разметку. Разметку кратности большей, чем единица назовем параллельной. Набор независимых данных, соответствующий такой разметке, будем называть параллельным списком. При разметке кратности 1 имеем частный случай параллельного списка, вырожденного до одноэлементного параллельного списка или просто элемента.

Следует отметить, что поддержка разметкой дуги нескольких независимых наборов данных позволяет описывать на уровне МВ массовый параллелизм. При этом инициализация вычислений в вершине может начинаться до формирования полной разметки, так как обработка каждого из входных наборов осуществляется независимо.

Например, если вершина V_i получает данные с дуги A_j с кратностью разметки **N**, то для формирования неполной разметки на выходной дуге A_k достаточно появления хотя бы одного набора данных r_m. Дальнейшее формирование разметки на входной дуге позволяет пополнять разметку на выходе. Данный механизм поддерживается аксиомами языка. Необходимым условием является наличие полных разметок только при окончании вычислений функции.

Назовем разметку дуги, не сформированную до конца, неполной.

Отметим также, что произвольное поступление элементов вектора значений **R** на обработку не приводит к неоднозначности, так как каждый элемент идентифицирован уникальным порядковым номером от **1** до **N**.

1.2. Описание программо-формирующих операторов

Для графического представления операторов используются специальные обозначения, приведенные на рис. 1.1 - 1.6. Наряду с графическими обозначениями применяется и текстовое описание элементов модели, аналогичное принятому в разработанном языке программирования.

Оператор интерпретации описывает функциональные преобразования аргумента. Он имеет два входа, на которые, через информационные дуги, поступают функция **F** и аргумент **X** (рис. 1.1).

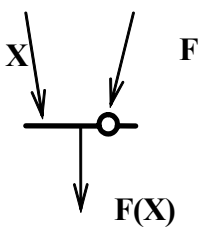

Рис. 1.1. Оператор интерпретации
с входами аргумента **X** и функции **F**.

Аргумент и функция могут являться результатами предшествующих вычислений.

Оператор интерпретации запускается по готовности данных, что фиксируется появлением разметки на входных дугах. Получение результата задается разметкой выходной дуги. При текстовом описании оператор интерпретации имеет две формы: постфиксную, обозначаемую знаком "**:**", и префиксную, при которой функция отделяется от аргумента знаком "^". Наличие двух способов записи одного оператора позволяет в дальнейшем комбинировать их с целью получения более наглядного текста программы. Следовательно, выражение **F(X)** оператор интерпретации позволяет задать одной из форм:

X:F или **F^X**.

Константный оператор не имеет входов (рис. 1.2). У него есть только один выход, на котором постоянно находится разметка, определяющая предписанное значение. Множество константных операторов информационного графа определяют внутреннюю начальную разметку МВ. В языковом представлении константный программо-формирующий оператор задается значением соответствующего типа.

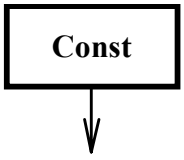

Рис. 1.2. Константный оператор

Оператор копирования (рис. 1.3) осуществляет передачу данных с одного своего входа на множество выходов. В графическом представлении данная передача фиксируется установкой разметки на дугах, связанных с выходами вершины при размеченной входной дуге. В текстовой форме операция копирования определяется через именование передаваемое величины и дальнейшее использование введенного обозначения в требуемых точках в качестве одной и той же связи.

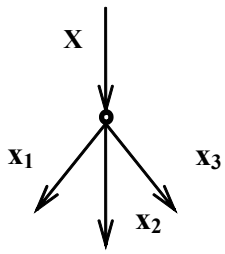

Рис. 1.3. Оператор копирования данных

Используется постфиксное именование размножаемого объекта в форме:

величина >> имя ,

и его префиксный эквивалент, имеющий вид:

имя << величина .

Например: **y << F^x; (x,y):+ >> c;**

Оператор группировки в список (рис. 1.4) имеет несколько входов и один выход. Он обеспечивает структуризацию и упорядочение данных, поступающих по дугам из различных источников. Порядок элементов определяется номерами входов, каждому из которых соответствует натуральное число в диапазоне от **1** до **N**. В текстовом виде оператор задается ограничением элементов списка круглыми скобками "(" и ")". Например:

(x1, x2, x3, x4).

Нумерация элементов списка в данном случае задается неявно в соответствии с порядком их следования слева направо (это же соглашение предполагается и в графическом представлении при отсутствии явной нумерации входов).

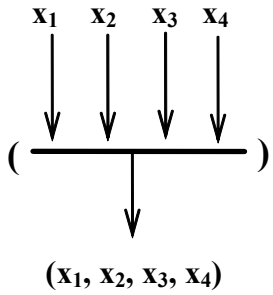

Рис 1.4. Оператор группировки в список

Выполняясь над дугами с разметкой любой кратности, оператор группировки в список на выходе всегда формирует разметку кратности 1. Количество элементов в сформированном списке равно сумме кратностей разметок всех входных дуг.

Наряду с группировкой в список данных предполагается возможность создания параллельных списков. **Оператор создания параллельного списка** (рис. 1.5) обеспечивает группирование элементов, аналогичное списку данных. Однако, кратность разметки, определяющая выходной набор данных равна сумме кратностей разметок всех входных дуг.

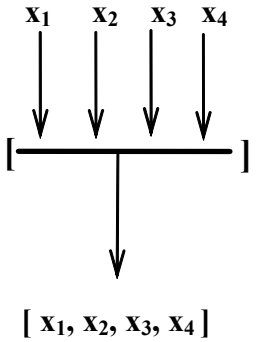

Рис 1.5. Оператор группировки в
параллельный список

Сформированная структура при выполнении оператора интерпретации рассматривается в другом контексте. В соответствии с ниже описанной алгеброй преобразований языка считается, что задаваемая функция использует каждый элемент данного списка как независимый аргумент. Если же параллельный список определяет набор функций, то все они выполняются одновременно над одним и тем же аргументом. Таким образом, данная конструкция обеспечивает организацию массового параллелизма. В текстовом виде группировка в параллельный список задается ограничением его элементов квадратными скобками "[" и "]". Например:

$$[x_1, x_2, x_3, x_4] .$$

При этом осуществляется сквозная перенумерация всех элементов сформированного списка слева направо с учетом суммарной кратности.

Оператор группировки в задержанный список (рис. 1.6) задается вершиной, содержащей допустимый вычисляемый подграф, в котором возможно несколько входов и выходов. Входы связаны с дугами, определяющими поступление аргументов, а выход

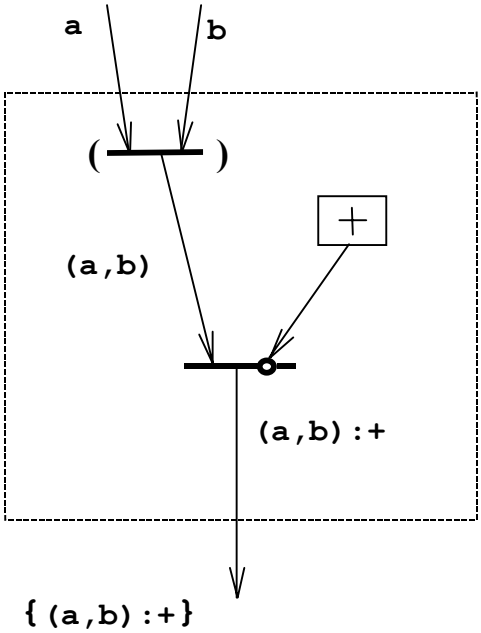

Рис 1.6. Список задержанных вычислений

определяет выдаваемый из подграфа результат. Специфической особенностью такой группировки является то, что ограниченные оператором задержки вершины (на графе ограничение задается контуром формируемой мультивершины), представляющие другие программо-формирующие операторы, не могут выполняться, даже при наличии всех аргументов. Их активизация возможна только при снятии задержки (раскрытии контура), когда ограниченный подграф становится частью всего вычисляемого графа.

Задержанный подграф создает на своем единственном выходе константную разметку, которая является образом (иконкой) данного подграфа. Эта разметка распространяется по дугам графа от одного программо-формирующего оператора к другому, размножаясь, входя в различные списки и выделяясь из них до тех пор, пока не поступит на один из входов не задержанного оператора интерпретации. При наличии на его обоих входах готовых данных происходит подстановка, вместо иконки, ранее определенного задержанного графа с сохранением входных связей. Опоясывающий подграф контур оператора задержки при этом «убирается», и происходит выполнение активированных операторов. В результате на выходной дуге раскрытого подграфа вновь формируется результирующая разметка, которая и является одним из аргументов оператора интерпретации, раскрывшего задержанный подграф. Данная процедура называется раскрытием задержанного подграфа. В текстовом виде группирование в список задержанных вычислений (для краткости будем называть также задержанный список) задается охватом соответствующих операторов фигурными скобками "{" и "}". Например:

$\{x_1, x_2, x_3, x_4\}$ или {(a,b):+} .

Наличие этой конструкции позволяет откладывать момент начала некоторых вычислений или вообще не начинать их, что необходимо при организации выборочной обработки данных.

1.3. Описание динамики функционирования

Правила распространения разметки по графу складываются из общих правил межоператорных переходов, правил срабатывания программо-формирующих операторов,

правил выполнения оператора интерпретации над предопределенными функциями модели (а далее и языка), правил эквивалентных преобразований операторов и связей допустимого графа (алгебры преобразований, связанной с исполнением отдельных операторов и описанной выше).

Правила межоператорных переходов задают распространение разметки по графу:

1. Если входные дуги вершины имеют разметку, то на выходных дугах происходит формирование разметки в соответствии с правилами срабатывания вершины, определяющий программо-формирующий оператор.

2. Если входные разметки имеют кратность, превышающую единицу, то для заданной вершины формирование выходной разметки может начинаться при неполной входной разметке независимо для каждого из сформированных наборов входных данных, и осуществляется в соответствии аксиомами срабатывания программо-формирующих операторов.

3. В процессе распространения разметка не убирается и не замещается. Каждая дуга графа может получить разметку только один раз. Из требования о недопустимости повторной разметки вытекает требование ацикличности графа.

4. Процесс распространения разметки заканчивается, когда все дуги графа имеют полную разметку в соответствии с предписанной кратностью или при невозможности распространения разметки.

Правила срабатывания программо-формирующих операторов конкретизируют формирование разметок на выходных дугах для каждого из ранее введенных операторов.

Оператор интерпретации обеспечивает преобразование входного набора данных **X**, выступающего в качестве аргумента, в выходной набор **Y**, играющего роль результата, используя при этом входной набор **F** в качестве функции, определяющей алгоритм преобразования. В постфиксной нотации, выбранной для дальнейших иллюстраций, данное преобразование можно записать следующим образом:

$$\mathbf{X:F} \Rightarrow \mathbf{Y}.$$

Можно рассмотреть множество унарных функций **F**, разделив его при этом на два подмножества:

$$\mathbf{F} = \{\mathbf{f}_1, \mathbf{f}_2\},$$

где $\mathbf{f}_1$ - множество предопределенных функций языка, для каждой из которых аксиоматически задается области определения и изменения; $\mathbf{f}_2$ - множество функций, порождаемых при программировании. Необходимо отметить, что областью определения любой функции из **F** является множество одноэлементных наборов данных. Обработка же параллельного списка определяется с помощью правил эквивалентных преобразований. Результатом выполнения функции может быть любой тип данных, включая параллельный список произвольной кратности. Следует отметить, что выбор базового набора предопределенных функций осуществляется в некоторой степени субъективно, исходя из соображений удобства пользования разрабатываемым языком. Вводятся аксиоматически определенные арифметические функции, функции сравнения и прочие, аналогично тому, как это сделано и в других языках программирования. Например, функция сложения двух чисел $\mathbf{x}_1, \mathbf{x}_2$, порождающая в качестве разметки число **y**, задается следующим образом:

$$(\mathbf{x}_1, \mathbf{x}_2)\mathbf{:+} \Rightarrow \mathbf{y},$$

где первый аргумент оператора интерпретации является двухэлементным списком данных. Каждый элемент этого списка должен быть числом. Второй аргумент оператора интерпретации является функцией сложения, обозначенной значком "+". Результат функции сложения, значение **y**, является атомарным элементом.

Наряду с определением функций, присущих всем языкам программирования, целесообразно определить множество функций, нестандартных в традиционном понимании. Например, целое число может непосредственно интерпретироваться как функция выбора элемента списка:

$$(x_1, x_2, \ldots x_i, \ldots x_n):i \Rightarrow x_i ,$$

где **i** - натуральное число, x_i – элемент списка. Данная функция выделяет из списка данных **i**-й элемент, который и определяет разметку выходной дуги.

Другой полезной предопределенной функцией является:

$$(b_1, b_2, b_3, \ldots b_n):? \Rightarrow [i_1, i_2, \ldots i_k] ,$$

где $(b_1,\ldots b_n)$ - список булевских величин; $[i_1,\ldots i_k]$ - параллельный список натуральных чисел, определяющих номера тех компонент булевского списка, которые имеют истинные значения. Наличие данной функции позволяет формировать условия, обеспечивающие выполнение нескольких альтернативных ветвей программы.

Наряду с определением операции интерпретации для аксиоматически определенных функций, она также определяется и для уже существующих программо-формирующих операторов. Так, в частности, определены следующие правила раскрытия задержанного списка:

$$\{x_1,\ldots,x_n\}:f \Rightarrow [x_1,\ldots,x_n]:f \quad (1.1),$$

$$x:\{f_1,\ldots,f_k\} \Rightarrow x:[f_1,\ldots,f_k] \quad (1.2),$$

$$\{x_1,\ldots,x_n\}:\{f_1,\ldots,f_k\} \Rightarrow [x_1,\ldots,x_n]:[f_1,\ldots,f_k] \quad (1.3).$$

Выражение (1.1) показывает, что, при наличии разметки на дуге, определяющей вход **f**, задержанный список данных $\{x_1,\ldots,x_n\}$ преобразуется в параллельный. Далее, если $x_1,\ldots,x_n$ являются допустимыми подграфами, следует получение их значений, после чего осуществляется выполнение заданного оператора интерпретации. Выражение (1.2) описывает аналогичное раскрытие задержанного списка функций при появлении разметки на входе, определяющем **x**. Если же оба аргумента оператора интерпретации являются задержанными списками (1.3), то они в начале воспринимаются, как константные значения, что определяет их немедленное преобразование в параллельные списки. После этого каждый список вычисляется, что приводит к разметке входных дуг описанного оператора интерпретации и его выполнению. Следует заметить, что вид операторов интерпретации, приведенный в выражениях 1.1-1.3, не является окончательным перед их вычислениями. Необходимо еще провести дополнительное приведение к элементарным вычислительным действиям в соответствии с правилами эквивалентных преобразований.

1.4. Эквивалентные преобразования

Правила эквивалентных преобразований операторов и связей допустимого графа определяют алгебру модели и языка программирования. Они позволяют осуществить трансформацию графа, обеспечивающую сведение сложных структурированных операций к набору элементарных действий над предопределенными компонентами. Возможна также обратная структуризация элементарных действий, полезная при адаптации полученной функциональной параллельной программы к архитектуре конкретной ВС. Эквивалентные преобразования определены на множестве программо-формирующих операций и отражают общие алгебраические свойства модели. Проведение этих преобразований может происходить как перед началом вычислений, когда они применяются к исходному информационному графу, так и непосредственно в ходе выполнения программы. В этом случае преобразования проходят на частично размеченном графе.

Для описания правил эквивалентных преобразований введем ряд обозначений. Отдельные значения элементов (значения одноэлементных параллельных списков) будем обозначать малыми латинскими буквами: **a, b, c, d, e, x, y, z** для данных и **f, g, h** для функций. Значения параллельных списков данных и функций обозначим соответствующими заглавными латинскими буквами. Аналогично для еще не вычисленных элементов будем использовать малые латинские буквы: **q, r, s, t, v, w**. Не вычисленные выражения внутри параллельных списков обозначим через эти же, но

заглавные буквы. Для описания порядкового расположения элементов списков будем использовать индексы, задаваемые натуральными числами и буквами **i, j, k, l, m, n**.

Например:

$$X = x_1, x_2, ..., x_i, ..., x_n \ ;$$

$$F = f_1, f_2, f_3 \ ;$$

$$W = w_1, ..., w_k \ .$$

Перед выполнением любой операции интерпретации выполняются (в случае необходимости) следующие эквивалентные преобразования.

1.4.1. Распространение ошибки

Аргументы операции интерпретации, содержащие одну или более фишек ошибки, проходят через операцию с формированием обобщенной ошибки:

$$(BOUNDERROR, 2.34):f \Rightarrow (ERROR\ ,\ (BOUNDERROR, 2.34))$$

1.4.2. Использование пустых элементов

В языке и модели допускается вырожденная форма, данных, называемая пустым элементом. Он обозначается **"."**. Допустимы следующие формы пустых списков:

(.) - пустой список данных;

[.] - пустой параллельный список; (1.4)

{.} - пустой задержанный список.

Список данных «**(.)**», или пустой список используется во многих языках программирования. Его длина равна 0. Пустой параллельный список эквивалентен пустому элементу:

$$. \equiv [.]$$

Пустой задержанный список «**(.)**» может временной защиты пустого элемента от преобразований, чтобы впоследствии его можно было использовать в качестве аргумента операции интерпретации. Например:

$$(\{.\}, b_2):1:F \Rightarrow \{.\}:F \Rightarrow .:F$$

Пустой элемент может использоваться в качестве аргумента и функции в операторе интерпретации. Использование в качестве функции эквивалентно отсутствию преобразований:

$$X:. \equiv X$$

Помимо этого использование пустого элемента в качестве функции позволяет раскрывать задержанные списки без выполнения функциональных преобразований:

$$\{X\}:. \Rightarrow X$$

Использование же пустого элемента в качестве аргумента оператора интерпретации в форме **".:f"** применяется при выполнении функции без параметров и для раскрытия задержанной функции:

$$.:\{F\} \Rightarrow .:F$$

Если среди элементов списка встречаются отдельные пустые элементы, то они удаляются из списка до выполнения его дальнейших преобразований:

$$(x_1, x_2, ., x_3, x_4, x_5, ., ., x_6) \equiv (x_1, x_2, x_3, x_4, x_5, x_6)$$

При этом длина списка уменьшается. Исключением является только пустой список, который не преобразуется. Аналогичным образом осуществляется «сжатие» любого подсписка данных, размещенного внутри списка. При этом за удаление пустых элементов

отвечает тот список данных, в котором эти элементы имеются:

$$(x_1,x_2,(.,x_3),(x_4,(x_5,.)),.,x_6) \equiv (x_1,x_2,(x_3),(x_4,(x_5)),x_6)$$

Если в списке данных имеется задержанный пустой список, то он не изменяется. Например:

$$(x_1,x_2,\{.\},x_3,\ x_4,x_5,.,\{.\},x_6) \equiv (x_1,x_2,\{.\},x_3,x_4,x_5,\{.\},x_6)$$

1.4.3. Раскрытие параллельных подсписков в списке данных

Пусть

$$X=[x_1,...x_n],\ Y=[y_1,...\ y_m],$$

тогда:

$$(X,\ Y) \equiv (x_1,...x_n,\ y_1,...\ y_m).$$

Данное правило показывает, что объединение двух параллельных списков внутри списка данных приводит к формированию единого списка, число элементов которого равно сумме элементов исходных параллельных списков. В общем случае количество списков может быть произвольным, что уже отмечалось раньше. Помимо этого параллельные списки могут быть вложены друг в друга произвольным образом:

$$(x_1,x_2,[x_3],[x_4,[x_5]],x_6) \equiv (x_1,x_2,x_3,x_4,x_5,x_6)$$

Таким образом, список данных обеспечивает преобразование своей внутренней структуры к более простому виду за счет избавления от пустых элементов и раскрытия вложенных в него параллельных списков. Первое позволяет уменьшить его размер, а второе обеспечивает линейность и наращивание элементов. Возможно одновременное использование обоих вариантов преобразования списка данных:

$$(x_1,x_2,[.,x_3],[x_4,[x_5,.]],.,x_6) \equiv (x_1,x_2,.,x_3,x_4,x_5,.,.,x_6) \equiv (x_1,x_2,x_3,x_4,x_5,x_6)$$

Подобные преобразования списка данных возможны из-за использования его в качестве синхронизирующего элемента. Список данных может использоваться в качестве операнда операции интерпретации или элемента других списков только в том случае, когда будут получены все его внутренние элементы и проведены необходимые преобразования внутренней структуры.

Параллельные списки не обеспечивают подобных преобразований из-за использования их в качестве носителей независимых наборов данных, которые могут обрабатываться на различных процессорах, в произвольном порядке и никоим образом должны быть связаны между собой. Вычисления над параллельным списком могут внутри вновь порождать параллельные списки, которые, однако, не могут быть объединены:

$$[X,\ Y] \equiv [[x_1,...x_n],\ [y_1,...\ y_m]\ ].$$

Отсутствие внутренних преобразований, но по иной причине, свойственно и задержанным спискам: в них вообще невозможны никакие вычисления до раскрытия.

Зачастую квадратные скобки параллельного списка играют роль условного ограничителя, они используются для сохранения приоритета операций, не нарушая при этом принятого порядка нумерации элементов. Исходя из сказанного, можно не ставить квадратные скобки там, где это не является необходимым. Например, одноэлементный параллельный список $[a] \equiv a$. Квадратные скобки можно не ставить и при наличии других типов охватывающих скобок:

$$([x_1,\ ...\ x_n]) \equiv (x_1,\ ...\ x_n)\ ,$$

$$[[x_1,\ ...\ x_n]] \equiv [x_1,\ ...\ x_n]\ ,$$

$$\{[x_1,\ ...\ x_n]\} \equiv \{x_1,\ ...\ x_n\}\ .$$

1.4.4. Раскрытие задержанных списков

При попытке интерпретации задержанного списка (выступает он в качестве аргумента или функции операции интерпретации – не важно), происходит преобразование в параллельный список, элементами которого становятся результаты вычисления выражений – элементов задержанного списка.

$$\{x_1,x_2,..,x_n\}:F \equiv [x_1,x_2,..,x_n]:F$$

$$X:\{f_1,f_2,..,f_n\} \equiv X:[f_1,f_2,..,f_n]$$

$$\{x_1,x_2,..,x_n\}:\{f_1,f_2,..,f_n\} \equiv [x_1,x_2,..,x_n]:[f_1,f_2,..,f_n]$$

Многократно вложенные задержанные списки полностью раскрываются оператором интерпретации и эквивалентны задержанному списку с одним уровнем вложенности:

$$\{\{X\}\} \equiv \{X\}$$

Поэтому:

$$\{\{\{\{X\}\}\}\}:. \Rightarrow X$$

Примеры:

$$\{1,4,7\}:- \Rightarrow [1,4,7]:- \Rightarrow [-1,-4,-7]$$

$$\{((2,3):+,45),(3,(3.14,2):*)\}:+ \Rightarrow$$
$$[((2,3):+,45),(3,(3.14,2):*)]:+ \Rightarrow$$
$$[(5,45),(3,6.28)]:+ \Rightarrow [50,9.280001e+000]$$

$$(2,3):\{+,-,/,* ,\%\} \Rightarrow (2,3):[+,-,/,*,\%] \Rightarrow$$
$$\Rightarrow [5,-1,6.666667e-001,6,(0,2)]$$

1.5. Правила интерпретации списков

В случае, когда справа от знака постфиксной операции интерпретации стоит список функций (параллельный или последовательный) или слева стоит параллельный список аргументов, применяются следующие правила интерпретации в том порядке, как они перечислены ниже:

1.5.1. Перенос круглых скобок со списка функций на результат операции интерпретации

Последовательный список функций (в том числе и одноэлементный) над некоторым аргументом эквивалентен операции интерпретации аргумента параллельным списком функций, результат которой становится элементом последовательного списка:

$$X:(f_1,f_2,...,f_3) \equiv (X:[f_1,f_2,...,f_3])$$

Применение пустого списка в качестве функции позволяет охватить скобками параллельный список без выполнения каких либо преобразований:

$$X:(.) \equiv (X) \quad .$$

Следует отметить, что введение обозначений (1.4) для различных пустых списков вместо обычных пар скобок связано с тем, что последние рассматриваются как специальные символы языка, за которыми закреплены предопределенные функции, несущие семантику, отличную от той, которую, в качестве функциональных аргументов операции интерпретации, имеют пустые списки. Данные функции участвуют в эквивалентных преобразованиях наравне с другими предопределенными и описываемыми функциями. Например, функция `"[]"` предназначена для раскрытия списков данных:

$$(X):[] \Rightarrow X \ .$$

Поэтому,

$$[x_1, \ldots, x_n]:[] \Rightarrow x_1:[], \ldots, x_n:[].$$

Функция "( )" используется для охвата элемента скобками и не эквивалентна "(.)":

$$X:() \equiv [x_1,\ldots,x_n]:() \equiv [x_1:(),\ldots,x_n:()] \neq X:(.) \ .$$

Совпадение действия данных функций происходит только на одноэлементном аргументе:

$$x:() \equiv x:(.) \equiv (x).$$

1.5.2. Интерпретация параллельных списков

Изначально выполнение функций оператором интерпретации определяется над отдельными элементами. Обработка параллельных списков сводится к множеству отдельных операторов интерпретации:

Параллельный список функций над некоторым аргументом эквивалентна параллельному списку операций интерпретации для каждой из функций над одним и тем же аргументом:

$$X:[f_1,f_2,\ldots,f_n] \equiv [X:f_1,X:f_2,\ldots,X:f_n]$$

Функция над параллельным списком аргументов эквивалентна параллельному списку операций интерпретации для этой функции над каждым из аргументов:

$$[x_1,x_2,\ldots,x_n]:f \equiv [x_1:f,x_2:f,\ldots,x_n:f]$$

Эти выражения показывают, каким образом кратную разметку некоторой дуги можно представить эквивалентной одноэлементной разметкой множества дуг. В более общем случае, когда разметка с кратностью больше единицы имеется на обеих входных дугах оператора интерпретации (то есть, как для функций, так и для аргументов), на выходе формируется параллельный список кратностью, равной произведению кратностей разметок его входных дуг. Порядок следования элементов сформированного параллельного списка определен таким образом, что данные выступают в качестве первичного ключа независимо от префиксной или постфиксной формы записи:

$$[x_1,x_2,\ldots,x_m]:[f_1,f_2,\ldots,f_n] \Rightarrow [x_1:[f_1,f_2,\ldots,f_n],$$
$$x_2:[f_1,f_2,\ldots,f_n],\ldots, x_m:[f_1,f_2,\ldots,f_n]] \Rightarrow$$
$$[[x_1:f_1,x_1:f_2,\ldots,x_1:f_n],$$
$$[x_2:f_1,x_2:f_2,\ldots,x_2:f_n],\ldots,[x_m:f_1,x_m:f_2,\ldots,x_m:f_n]] \equiv$$
$$\equiv [f_1,f_2,\ldots,f_n]\hat{}[x_1,x_2,\ldots,x_m]$$

Если в параллельном списке имеются вложенные параллельные списки, происходит рекурсивное распространение второго операнда до тех пор, пока не иссякнет вложенность параллельных списков. Пример для параллельного списка данных выглядит следующим образом:

$$[x_1,[x_2,[x_3]]]:f \equiv [x_1:f,[x_2:f,[x_3:f]]]$$

Аналогичным образом осуществляется привязка аргумента к вложенному параллельному списку функций:

$$X:[f_1,[f_2,[f_3]]] \equiv [X:f_1,[X:f_2,[X:f_3]]]$$

Если вложенными параллельными списками являются аргумент и функция оператора интерпретации, то происходит рекурсивное пересечение всех возможных вариантов. В качестве первичного ключа, определяющего порядок следования элементов, выступает список функций:

$$[x_1,[x_2,[x_3]]]:[f_1,[f_2,[f_3]]] \Rightarrow$$
$$[[x_1,[x_2,[x_3]]]:f_1, [x_1,[x_2,[x_3]]]:[f_2,[f_3]]] \Rightarrow$$

$$[[x_1,[x_2,[x_3]]]:f_1,\ [[x_1,[x_2,[x_3]]]:f_2,\ [x_1,[x_2,[x_3]]]:[f_3]]] \Rightarrow$$
$$[[x_1,[x_2,[x_3]]]:f_1,\ [[x_1,[x_2,[x_3]]]:f_2,\ [x_1,[x_2,[x_3]]]:[f_3]]] \Rightarrow$$
$$\ldots \Rightarrow$$
$$[[x_1:f_1,[x_2:f_1,[x_3:f_1]]],\ [[x_1:f_2,[x_2:f_2,[x_3:f_2]]]:f_2],$$
$$[x_1:f_3,[x_2:f_3,[x_3:f_3]]]]$$

Правила эквивалентных преобразований и интерпретации списков тесно взаимосвязаны с набором предопределенных функций. Их комбинации определяют широкие возможности по формированию структуры функциональной параллельной программы. Эта тема более подробно рассматривается при описании языка.