УДК 004.4'2

ИНСТРУМЕНТАЛЬНАЯ ПОДДЕРЖКА ГРАФИЧЕСКОЙ ВЕРСИИ ЯЗЫКА ФУНКЦИОНАЛЬНО-ПОТОКОВОГО ПРОГРАММИРОВАНИЯ «ПИФАГОР»

Гризан С.А.
Научный руководитель — д.т.н., профессор Легалов А.И.
Сибирский федеральный университет

Визуальное программирование можно упрощенно описать как создание программы путем манипулирования графическими объектами вместо написания ее текста. Наиболее полно сформулированным определением визуального программирования считается определение в работах Маргаретт Барнет. «Визуальное программирование – это программирование, в котором для передачи семантики используется более чем одно измерение. В качестве примеров таких дополнительных измерений можно привести применение многомерных объектов, пространственных или временных отношений для спецификации семантики отношения «до–после»». Традиционная (языковая) техника программирования с семантической точки зрения одномерна, потому как семантика программы определяется «линией» – строкой текста, содержащего собственно программу.

Особенно хорошо такой метод показал себя для языков потока данных, так как написанные с их помощью программы описывают направленный граф, а потому работа с этим графом на уровне графических примитивов выглядит более логичной, чем его представление в текстовом виде.

Так как еще на начальных этапах разработки функционально-потокового языка «Пифагор» были введены графические обозначения, позволяющие представить в двумерном виде информационный граф программы, целесообразна разработка визуальной среды разработки, раскрывающей достоинства графического языка программирования, основанного на потоках данных.

Каждому оператору языка ставится в соответствие графическое представление, помещающееся в узлы информационного графа.

Рис. 1. Вход функции

Рис. 2. Выход функции

Вход функции обозначает точку начала выполнения функции, с которой начинается распространение потоков данных. На выходную информационную связь подается передаваемый в функцию аргумент. На единственную связь выхода функции подается возвращаемый функцией результат.

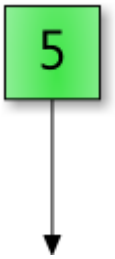

Рис. 3. Константа

Рис. 4. Внешняя функция

Одним из основополагающих примитивов является значащая величина (константа). На единственную выходную связь подается значение, указанное в константе.

Другим примитивом, по сути схожим с константой, является внешняя функция. Определенная в пределах зоны видимости функция с указанным именем подается на единственную выходную связь.

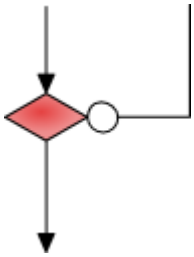

Рис. 5. Оператор интерпретации

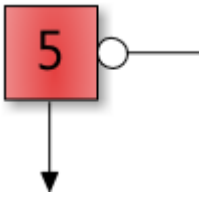

Рис. 6. Оператор интерпретации с внесенным аргументом

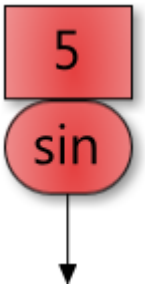

Рис. 7. Оператор интерпретации с внесенными аргументом и функцией

Рис. 8. Оператор интерпретации с внесенной функцией

Представление оператора интерпретации имеет 2 входа и 1 выход. Один из входов помечается стрелкой стандартного вида и обозначает информационную связь с аргументом функции. Подходящая ко второму входу информационная связь с функцией имеет на конце окружность вместо стрелки для лучшего различения типов входов оператора. Единственный выход оператора формирует информационную связь с результатом применения функции над аргументом.

Помимо полной формы записи возможны сокращенные варианты, позволяющие заносить постоянные входные параметры в оператор. Так, может быть внесен аргумент, передаваемый в функцию, в случае если он выражен константной. При этом у оператора остается 1 вход для применяемой функции, также отмеченный стрелкой с окружностью на конце, и 1 выход для результата. Форма оператора согласована с формой константы в то время, как его цвет унаследован от оператора интерпретации.

Часто возникает необходимость применить к данным именованную внешнюю функцию, для чего введен соответствующий сокращенный вариант оператора интерпретации. Здесь редуцирована информационная связь с применяемой функцией, кото-

рая записана внутри оператора. Форма в данном случае также связана с формой внешней функции, а цвет говорит о принадлежности к группе операторов интерпретации.

Наиболее сокращенный вариант не содержит входов. Как аргумент, так и функция являются постоянными и записываются в самом операторе. На единственный выход подается результат. В представлении сохраняется логика выбора цвета и формы, спроецированная на еще большую степень редукции.

Оператор копирования дублирует поток данных, поступивший на его вход, на все его выходы. В текстовом представлении такая задача обычно решается введением идентификаторов, которые во многих случаях излишни в визуальном представлении.

Вокруг сгруппированных в задержанный список операторов изображается пунктирная рамка, а фон внутри неё закрашивается полупрозрачным фоном. При формировании вложенных задержанных списков, цвет фона становится темнее с ростом уровня включения.

Рис. 9. Оператор копирования

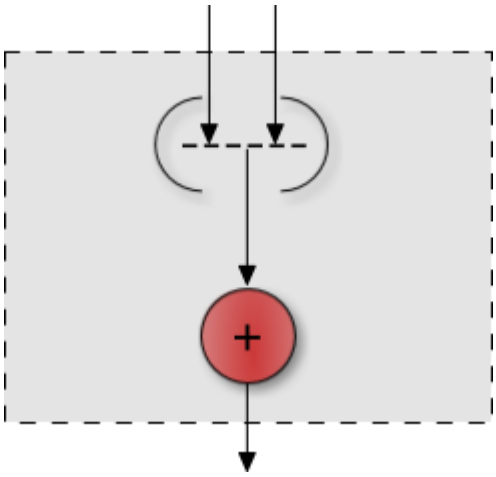

Рис. 10. Группировка в задержанный список

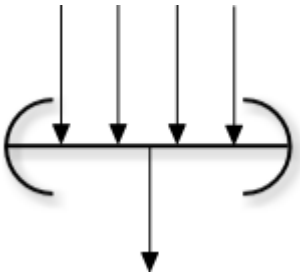

Рис. 11. Оператор группировки в список

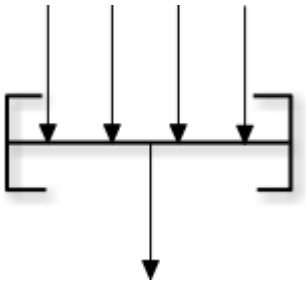

Рис. 12. Оператор группировки в параллельный список

Оператор группировки в список формирует список из данных, поступивших в нумерованные входы, и подает его на выходную информационную связь.

Аналогично предыдущему оператору, оператор группировки в параллельный список формирует параллельный список из поступивших данных.

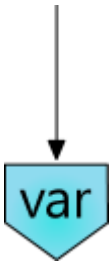

Рис. 13. Назначение идентификатора

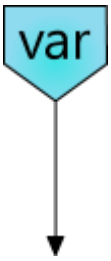

Рис. 14. Использование идентификатора

С помощью назначения идентификатора можно именовать информационную связь для использования данных в другом месте информационного графа без необходимости добавления перекрещивающихся связей. Данный примитив, как и метки в большинстве языков, рекомендуется к использованию только при большой отдаленности соединяемых операторов, которую нельзя разрешить переформированием структуры графа, ввиду неочевидности результата, или в случае, если формируемый информационный граф неплоский, так как его применение сильно ухудшает читаемость алгоритма.

Введенные визуальные представления лучше всего проиллюстрировать при помощи примера программы, суммирующей элементы вектора. В текстовом виде она выглядит следующим образом:

```
// Функция, возвращающая сумму элементов вектора
VecSum << funcdef Param
{ Len << Param:|;
 Return << .^[((Len,2):[<,=,>]):?]^
 (   { Param:[] }, { Param:+ },
     { block
       {  OddVec << Param:[(1,Len,2):..];
          EvenVec << Param:[(2,Len,2):...];
          ([OddVec,EvenVec]:Vsum):+
          >>break
       } // конец блока
     } // конец задержанного списка
  ) }.
```

Соответствующее данной программе визуальное представление:

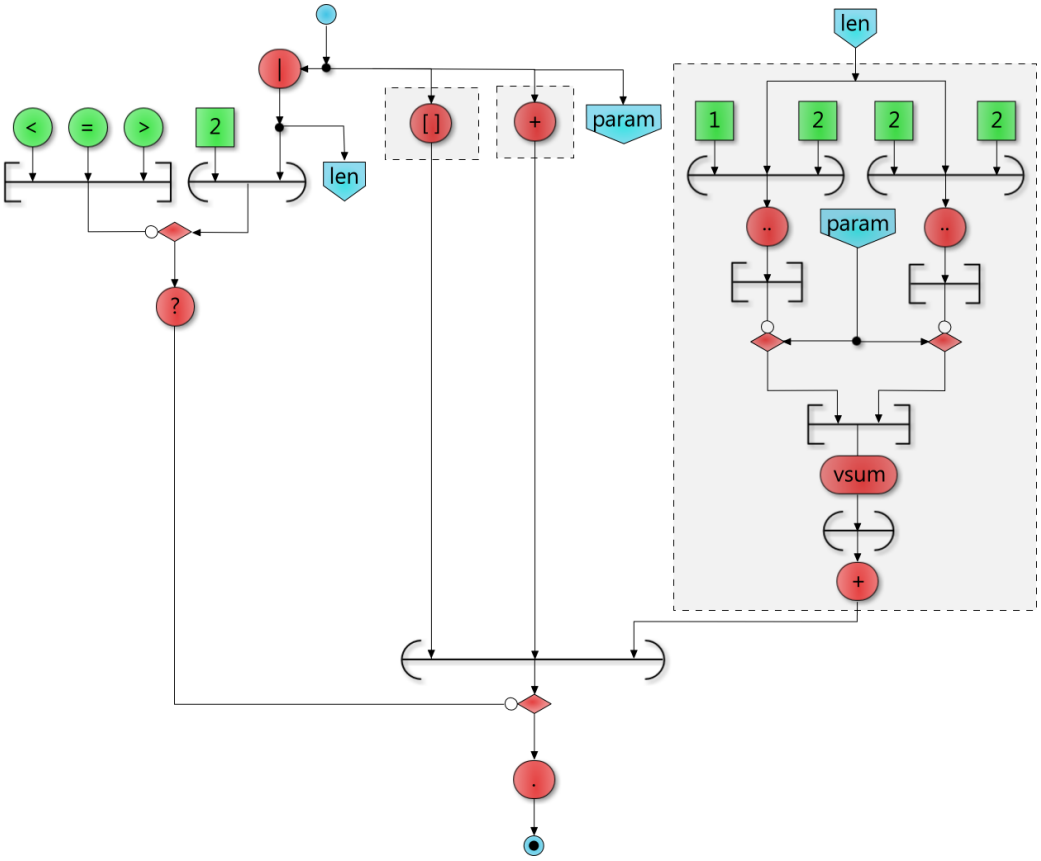

Рис. 15. Визуальное представление программы суммирования элементов вектора

Данные представления найдут применение в проектируемой визуальной среде разработки для языка «Пифагор», позволив разрабатывать программы, строя информационный граф её функций в графическом дизайнере.