Описание языка Algolang

Общие сведения

Algolang – это язык описания (разметки) информационных графов алгоритмов. Предназначен он для достаточно компактной и в то же время отображающей информационную структуру графа его записи, по которой затем можно построить визуализацию информационного графа. Algolang описывает граф на уровне вершин и дуг, но не в явном виде, а через логические выражения, определяющие их существование. Если описание составляется на основе программы, реализующей вычислительный алгоритм, то эта логика, как правило, схожа с логикой в этой программе.

Основные концепции языка

Algolang позволяет описывать информационную структуру фрагментов программ, относящихся к расширенному линейному классу (см. определение того, что такое линейный класс программ, в книге «Параллельные вычисления»). Расширение заключается в том, что неоднородные формы, задающие выражения переменных, границы изменения параметров циклов и условия передачи управления, не обязательно являются линейными.

1) Весь информационный граф представляется в виде набора опорных многогранников. Каждый многогранник описывается один раз, вне зависимости от того, для скольких операторов присваивания он является опорным.

2) Описание опорного многогранника состоит из двух частей. Сначала задаются границы изменения параметров его опорного цикла с указанием имени каждого параметра. Затем задаётся линейное пространство итераций (ЛПИ) и зависимости по данным (отдельно для каждой опорной области, входящей в ЛПИ) для каждого из операторов присваивания, имеющих многогранник в качестве опорного.

3) За задание ЛПИ отвечают «блоки операций». Их количество для каждого оператора присваивания (для каждой ф-ии указывается, к какому оператору присваивания она относится) определяет количество опорных областей и каждый включает в себя ф-ию, значение которой в каждой из целочисленных точек опорного многогранника определяет вхождение / невхождение этой точки в опорную область. В первом случае координаты целочисленной точки сохраняются в качестве текущих параметров блока и происходит его разбор

4) В каждый «блок операции» входит множество «блоков входных дуг», отвечающих за задание зависимостей по данным. В каждом «блоке входных дуг» указываются относительные (параметров «блока операции») либо абсолютные координаты некоторой целочисленной точки в произвольном опорном многограннике. Эта точка – источник входной дуги для точки, задаваемой значениями текущих параметров «блока операции».

Примеры описания информационных графов

Здесь приведено описание нескольких простых информационных графов, на основании которых можно сделать своё. Каждый пример состоит из двух частей: первая – фрагмент программы на C / C++ / Fortran вторая -–описание разметки информационного графа этого фрагмента. За этим

разделом следует формальное описание используемых здесь структур.

Пример 1:

C++
```
for (int i = 0; i < N; ++i) {
      for (int j = 0; j < N; ++j) {
            for (int k = 0; k < N; ++k) {
                  A[i][j][k] = A[i][j - 1][k] * A[1][j - 1][k];
            }
      }
}
```

Algolang
```
<algo>
   <params>
      <param name = "N" type = "int" value = "3"></param>
   </params>
   <block dims = "3">
      <arg name = "i" val = "1..N"></arg>
      <arg name = "j" val = "1..N"></arg>
      <arg name = "k" val = "1..N"></arg>

      <vertex condition = "" type = "1">
            <in src = "i, j - 1, k"></in>
            <in src = "1, j - 1, k"></in>
      </vertex>
   </block>
</algo>
```

Пример 2:

C++
```
for (int i = 0; i < N; ++i) {
      for (int j = 0; j < N; ++j) {
            for (int k = 0; k < N; ++k) {
                  A[i][j][k] = A[i - 1][j][k] * A[i][j - 1][k] +
                               A[i][j][k - 1];
            }
      }
}
for (int i = 0; i < N; ++i) {
      for (int j = 0; j < 2; ++j) {
            for (int k = 0; k < N; ++k) {
                  B[i][j][k] = A[i][j][k] + B[i - 1][j][k] * B[i][j - 1][k] +
                               B[i][j][k - 1];
            }
}
```

Algolang
```
<algo>
   <params>
      <param name = "N" type = "int" value = "6"></param>
   </params>

   <block id = "0" dims = "3">
      <arg name = "i" val = "1..N"></arg>
      <arg name = "j" val = "1..N"></arg>
      <arg name = "k" val = "1..N"></arg>

      <vertex condition = "" type = "1">
            <in src = "i - 1, j, k"></in>
            <in src = "i, j - 1, k"></in>
```

```
            <in src = "i, j, k - 1"></in>
      </vertex>
   </block>
   <block id = "1" dims = "3">
      <arg name = "i" val = "1..N"></arg>
      <arg name = "j" val = "1..2"></arg>
      <arg name = "k" val = "1..N"></arg>

      <vertex condition = "" type = "1">
            <in bsrc = "0" src = "i, j, k"></in>
            <in src = "i - 1, j, k"></in>
            <in src = "i, j - 1, k"></in>
            <in src = "i, j, k - 1"></in>
      </vertex>
   </block>
</algo>
```

Пример 3:

Fortran

```
subroutine STEP2(x,y,pov)
      complex x, y, pov, u, v
      u = x + y
      v = (x-y)*pov
      x = u
      y = v
      return
end

subroutine FFT2(X, POV, N, N2, L)
      C
      C L = Log2N
      C N2 = N/2
      C
      complex X(0:N-1), POV(0:N2-1)
      DO I = 0, L-1
      DO J = 0, N2/2**I-1
      DO K = 0, 2**I-1
            call STEP2(X(2*J*2**I+K)), X(2*J*2**I+2**I+K), POV(J*2**I))
      END DO
      END DO
      END DO
      return
end
```

Algolang

```
<algo>
      <params>
            <param name = "N" type = "int" value = "5"></param>
      </params>
      <block id = "0" dims = "2">
            <arg name = "I" val = "1..N"></arg>
            <arg name = "J" val = "1..2^N"></arg>

            <vertex condition = "J % (2^I) < 2^(I - 1)" type = "1">
                  <in src = "I - 1, J"></in>
                  <in src = "I - 1, J + 2^(I -1)"></in>
            </vertex>
            <vertex condition = "J % (2^I) >= 2^(I - 1)" type = "2">
                  <in src = "I - 1, J"></in>
                  <in src = "I - 1, J - 2^(I -1)"></in>
```

```
            </vertex>
        </block>
</algo>
```

Формальная структура описания информационного графа

В расширенной форме Бэкуса-Наура основные конструкции языка задаются следующим образом:

<Algorithm> ::== *<algo>* <Parameters> { <Block> } *</algo>*
<Parameters> ::== *<params>* { <Parameter> } *</params>*
<Parameter> ::== *<param name = “*<Name>*” type = “*<Type> [value = “<Number>*”]></param>*
<Type> ::== *int* | *float*
<Block> ::== *<block* [*id = “*<Number>*“*] *dims = “*<Number>*”*>{Argument}{Vertex}*</block>*
<Argument> ::== *<arg name = “*<Name>*” val = “*<RegExp>..<RegExp>*”></arg>*
<Vertex> ::== *<vertex condition = “*<RegExp>*” type = “*<Number>*”*>{Source}*</vertex>*
<Source> ::== *<in* [b*src = “*<Number>*“*] *src = “*<RegExp>*{,*<RegExp>*}”></in>*

Что скрывается за понятиями <Number>, <Name> и <RegExp>, а также зачем нужны опциональные (заключенные в []) выражения, поясним отдельно:

<Number> - это любое целое число. В теории. На практике та версия Algoview, которая используется при выполнении студенческого практикума, не воспринимает числа за пределами [-100, 100], для выполнения задания этого более чем достаточно.

<Name> - это любая ASCII-строка.

<RegExp> - это регулярное выражение. Для оценки значения регулярного выражения используется библиотека http://www.partow.net/programming/exprtk/ (https://github.com/ArashPartow/exprtk). Соответственно, разрешается всё, что она поддерживает, однако результат будет приведён либо к true/false (при вычислении condition), либо к целому числу (в ином случае). В качестве переменных в выражении могут выступать значения полей name у <Argument> и <Parameter>.

Перейдём к смыслу опциональных выражений.

Поле value у <Parameter> нужно для того, чтобы система могла сразу прочитать конкретное значение внешнего параметра и не запрашивала его у пользователя. Web-системе, используемой в задании, дополнительно запрашивать что-либо у пользователя запрещено, поэтому value в описании должно быть **всегда**. Значение параметра, где его нет, будет считаться равным 1.

Поле id у <Block> добавляет в <Block> параметр, по которому к нему можно обратиться. Если не задать, то будет равно -1

Поле bsrc у <Source> указывает, что зависимость по данным идёт из другого блока. В качестве значения указывается id блока-источника (если блока с таким id нет либо id == -1, то зависимость заведомо не строится).