Российская академия наук
Сибирское отделение
Институт систем информатики
им. А.П. Ершова

Д.С. Гордеев

Модель визуальной отладки Cloud Sisal программ с помощью цветных сетей Петри

Препринт

Новосибирск 2025

В статье описывается метод моделирования Cloud Sisal программ с помощью цветных сетей Петри специального вида с целью реализации визуальной отладки в рамках проекта по исследованию и разработке языка Cloud Sisal и инструментальных средств разработки программ с помощью данного языка. Проект в настоящее время реализуется в лаборатории конструирования и оптимизации программ Института систем информатики им. А.П. Ершова СО РАН. Cloud Sisal является функциональным языком программирования, предназначенным для научных вычислений. Язык развивается в рамках проекта по созданию системы поддержки параллельного программирования для облачных вычислений CPPS в научных и образовательных целях. В силу выбранных технологий реализации, развиваемый в настоящее время отладчик языка Cloud Sisal позволяет отлаживать программу средствами, доступными в большинстве используемых современных браузеров. При этом отладка программ на языке Cloud Sisal допустима на задачах малой размерности в рамках клиентского браузера, а отлаженные программы предназначены для исполнения на данных большой размерности с помощью супервычислителя. Программы на языке Cloud Sisal представимы в виде атрибутированных иерархических ориентированных ациклических графов с портами. В данной статье для визуальной отладки предложен метод моделирования семантики языка с помощью цветных сетей Петри, что позволяет использовать свойства сетей Петри для наглядной визуализации процесса отладки, а также для реализации такого известного метода отладки как точки останова, являющегося основным методом отладки в современных интегрированных средах разработки.

Siberian Division of the Russian
Academy of Sciences
A.P. Ershov Institute of Informatics Systems

D.S. Gordeev

Model of visual debugging of Cloud Sisal programs with colored Petri nets

Preprint

Novosibirsk 2025

The paper describes a method of modeling Cloud Sisal programs using colored Petri Nets of a special type. The aim is to implement visual debugging within the framework of the Cloud Sisal language research project, being carried out by the Laboratory of Program Design and Optimization, A.P. Ershov Institute of Informatics Systems, SB RAS. Cloud Sisal is a functional programming language designed for scientific computing. The language is being developed as part of a project intended to create the CPPS, a parallel programming support system for cloud computing for scientific and educational purposes. Thanks to the technologies selected, the currently developed Cloud Sisal language debugger allows its users to debug a program using the tools available in most modern browsers. Importantly, debugging programs in Cloud Sisal can be done on low-dimensional problems within the client browser, and the programs debugged are intended for execution on high-dimensional data using a supercomputer. Programs in the Cloud Sisal language are represented in the form of attributed hierarchical oriented acyclic graphs with ports. This paper suggests applying for visual debugging a method of modeling language semantics using colored Petri Nets. This makes it possible to use the properties of Petri Nets to visualize the debugging process, as well as to implement the well-known debugging method of breakpoints, which is the main debugging method for modern integrated development environments.

ВВЕДЕНИЕ

Целью проекта облачной системы параллельного программирования CPPS [1] является предоставление возможности решения вычислительноёмких задач с помощью доступных по сети супервычислителей широкому кругу пользователей без прямого доступа к вычислителям большой мощности. Пользователи создают и отлаживают программы на языке Cloud Sisal [2, 3] с помощью веб-приложения в рамках программы-браузера, а решение вычислительных задач осуществляется на поддерживаемом системой супервычислителе после предварительной адаптации с помощью настраиваемого компилятора из комплекта системы CPPS. При этом возникает задача поддержки процесса программирования и отладки Cloud Sisal программы на локальной системе пользователя перед запуском на супервычислителе. При трансляции программы на языке Cloud Sisal создаётся граф внутреннего представления IR входной программы, который представим в виде иерархического ориентированного графа [4]. Задача визуализации графа IR рассматривается в работе [5].

При отладке Cloud Sisal программ полезно иметь визуальное изображение процесса исполнения для понимания движения данных по графу, а также иметь возможность вмешиваться в процесс движения данных, например, с помощью известного метода точек останова, позволяющего останавливать вычисления, наблюдать и изменять промежуточные значения в остановленных вычислениях.

В данной работе обсуждается моделирование семантики выражений языка Cloud Sisal с помощью сетей Петри [6], позволяющее решать задачу визуальной отладки программ на языке функционального программирования Cloud Sisal в рамках проекта развития облачной системы параллельного программирования CPPS.

1.Внутреннее представление IR

Опишем структуру графа внутреннего представления. Вершины IR-графа соответствуют выражениям программы, порты соответствуют входным и выходным значениям выражений, а дуги отражают передачу данных между портами вершин. Каждому выражению соответствуют упорядоченные множества входных и выходных портов.

Входные порты соответствуют аргументам выражения, а выходные порты соответствуют его результатам. В силу свойства языка Cloud Sisal IR-граф является ациклическим [3]. Вершины соответствуют вычислениям со значениями во входных портах, результаты которых передаются в выходные порты вершин. Существует специальный вид вершин, обозначающих литералы (константы) различных типов, для которых не существует входных портов. Вершины бывают простыми и составными. Простые вершины не имеют внутренней структуры помимо ассоциированной с ними операции. Составные вершины соответствуют составным выражениям Cloud Sisal программы, таким как условные или циклические выражения, и дополнительно содержат упорядоченные множества вершин, соответствующих выражениям, из которых они состоят. Для каждого составного выражения количество, порядок и типы того множества вершин, которые непосредственно содержатся в нем, задаются типом (или семантикой) составного выражения. IR-графы являются атрибутированными. Каждый атрибут состоит из уникального имени атрибута и его значения. Атрибуты сопоставлены элементам (вершинам, дугам и портам) IR-графа и используются не только для выражения семантики Cloud Sisal программы, но и для ее визуальной обработки. Для составных вершин, например, именем атрибута может быть «тип вершины» со значениями «цикл», «функция» и другими, а также «цвет границы», «форма границы», «размер региона», «позиция» и так далее.

Текст Cloud Sisal программы транслируется во внутреннее представление IR, которое описывается как иерархический ориентированный граф с портами. Формально такой граф описывается следующим образом. Пусть задан граф,

$$G=\left(V,P,F^{v}_{p,i},F^{v}_{p,o},E\right) \quad (1)$$

где V это непустое конечное множество вершин, P - это непустое конечное упорядоченное множество портов, представимое в виде объединения двух непересекающихся множеств P_i и P_o, $F^{v}_{p,i}$ - это функция $F^{v}_{p,i}:P_i\to V$, $F^{v}_{p,o}$ - это функция $F^{v}_{p,o}:P_o\to V$, E это подмножество декартово произведения $(P_o\times P_i)$. Множество P_i называется множеством входных портов, а множество P_o называется множеством выходных портов. В случае, если функции $F^{v}_{p,i}$ и $F^{v}_{p,i}$ являются взаимно однозначными, то каждая вершина имеет ровно один входной порт и ровно один выходной, и граф G эквивалентен классическому графу $G'=(V',E')$, где E' есть подмножество декартова произведения $(V'\times V')$.

Для вершины v из V и дуги (p_o,p_i) из E будем говорить об инцидентности, если и только если $\left(F^{v}_{p,o}(p_o)=v\right)\vee\left(F^{v}_{p,i}(p_i)=v\right)$. Для вершин v_1 и v_2 из V будем говорить о смежности вершин, если и только если существует ребро (p_o,p_i) из E такое, что $\left(\left(F^{v}_{p,o}(p_o)=v_1\right)\wedge\left(F^{v}_{p,i}(p_i)=v_2\right)\right)$ или $\left(F^{v}_{p,o}(p_o)=v_2\right)\wedge\left(F^{v}_{p,i}(p_i)=v_1\right)$.

Каждая вершина графа внутреннего представления IR соответствует функции Cloud Sisal программы, по которой он построен. Входные порты P_i вершины v отвечают аргументам соответствующей функции, а её выходные порты P_o - результатам вычислений данной функции. Дуга (p_o,p_i) в графе

G обозначает передачу данных из выходного порта p_o во входной порт p_i.

При визуализации вершины графов обычно представляются геометрическими фигурами, такими как прямоугольники, окружности или более сложные многоугольники [5,7]. Дуги часто представлены в виде ломаных или гладких кривых. Порты, в зависимости от применения изображения графа, могут отображаться с помощью более мелких кругов или квадратов, причём обычно они располагаются на границах фигур, представляющих вершины. При записи решения задачи с помощью языка программирования часто функции вызывают другие функции. На графе внутреннего представления это отражается наличием вложенных графов, которые могут быть как связными, так и несвязными. В таких случаях принято считать, что вершина имеет атрибут, значением которого является вложенный граф, а саму вершину принято называть составной. По отношению существования вложенного графа образуется иерархия графов внутреннего представления. Также для определения стилей визуального отображения геометрических фигур требуется хранить информацию об атрибутах вершин и их значениях. Далее будем рассматривать графовую модель G с атрибутами. Пусть задан граф,

$$G=\left(V,P,F^{v}_{p,i},F^{v}_{p,o},E,N,M,A_v,A_p,A_e\right) \qquad (2)$$

где $V,P,F^{v}_{p,i},F^{v}_{p,o},E$ определяются аналогично определению (1), и N является конечным множество имён атрибутов, M является конечным множеством значений атрибутов, а отображение $A_P: P\times N\to M$ задаёт множество именованных атрибутов для вершин графа G. Аналогично A_p и A_e задают множества именованных атрибутов для портов и дуг графа G соответственно.

Моделирование с помощью сетей Петри

Рассмотрим далее типичные конструкции языка Cloud Sisal, такие как простые, условные, циклические и рекурсивные выражения, а также соответствующие им графы внутреннего представления IR. Для каждого случая построим сети Петри специального вида, которые далее будет называть CPN*.

Определение сетей Петри

Кратко напомним определение сетей Петри [6]. Сеть Петри PN - это (P, T, F, W, M_0), где P - конечное множество мест, T - конечное множество переходов, причем $P \cap T = \emptyset$, $F \subseteq P \times T \cup T \times P$, причём $\forall\, x \in P \cup T\ \exists\, y \in P \cup T : (x, y) \in F \vee (y, x) \in F$, W - функция кратности $W : F \to N \setminus \{0\}$, а M_0 - начальная разметка $M_0 : P \to N$, описывающая количество фишек в местах сети. Если места сети упорядочены, то каждому переходу $t \in T$ сопоставлены два вектора $t^I = (p_1^I, \ldots, p_n^I)$, где $\forall\, i\ \exists\, (x_i, t) \in F$, и $t^O = (p_1^O, \ldots, p_m^O)$, где $\forall\, i\ \exists\, (t, p_i^O) \in F$. Переход $t \in T$ готов к срабатыванию при некоторой разметке $M : P \to N$, если $\forall\, p \in W((p,t)) \leqslant M(p)$. Срабатывание перехода $t \in T$ на разметке M_1 порождает разметку M_2 по следующему правилу: $\forall\, p \in P : M_2(p) = M_1(p) - W((p,t)) + W((t,p))$. Сработать может любой переход, готовый к срабатыванию.

Определение специальных цветных сетей Петри

Поскольку в данной работе рассматривается моделирование вычислений функций, описываемых языком Cloud Sisal, рассмотрим класс сетей Петри, заданный следующим определением.

Определение. Цветная сеть Петри CPN* - это $(P, T, F, W, C, Q, F_T, M_0)$, где P - конечное множество мест,

T - конечное множество переходов, причем $P \cap T = \emptyset$, $F \subseteq P \times T \cup T \times P$, причём $\forall\, x \in P \cup T\ \exists\, y \in P \cup T : (x, y) \in F \vee (y, x) \in F$, W - функция кратности $W : F \to \{1\}$, C - множество цветов фишек, Q - отношение совпадения на множестве цветов, F_T - функция приписывающая каждому переходу $t \in T$ пару векторных функций $(f_T, \wp_T)$, где $f_T : R^k \to R^m \wedge \wp_T : R^k x C^k \to C^m \wedge k = |t^I| \wedge m = |t^O|$, где f_T может быть задана как формулой, так и некоторой сетью CPN*, а $M_0 : P \to R \times C$ - начальная разметка, задающая параметры моделируемой функции. В качестве фишек будем рассматривать пары (v, c), где v - это значение фишки, а c - это цвет значения фишки.

Множество цветов $C = \overset{\infty}{\underset{j=1}{U}} \mathbb{Z}^j$, а отношение совпадения Q задано по следующим правилам:

1) Если $c_1 = (z_1) \wedge c_2 = (z_2)$, то $(c_1, c_2) \in Q \Leftrightarrow (z_1 = z_2 \vee z_1 = 0 \vee z_2 = 0) \wedge z_1 \geq 0 \wedge z_2 \geq 0$.

2) Если $c_1 = (z_1^1, .., z_n^1) \wedge c_2 = (z_1^2, .., z_n^2)$ и $\overline{c}_1 = (z_1^1, .., z_{n-1}^1) \wedge \overline{c}_2 = (z_1^2, .., z_{n-1}^2)$, то $(c_1, c_2) \in Q \Leftrightarrow (\overline{c}_1, \overline{c}_2) \in Q \wedge ((z_n^1), (z_n^2)) \in Q$.

Отношение совпадения Q обладает следующими свойствами:

1) $\exists\, \widetilde{C} \subset C : \forall\, \tilde{c} \in \widetilde{C}\ \forall\, c \in C \setminus \widetilde{C} : (c, \tilde{c}) \notin Q$, то есть существуют несравнимые цвета.

2) $(c_1, c_2) \in Q \Rightarrow (c_2, c_1) \in Q$, то есть отношение симметрично.

3) $\forall\, c \in C : c \notin \widetilde{C} \Rightarrow (c, c) \in Q$.

4) $\exists\, c_1, c_2, c_3 \in C : (c_1, c_2) \in Q \wedge (c_2, c_3) \in Q \wedge (c_1, c_3) \notin Q$, например, $c_1 = 1, c_2 = 0, c_3 = 2$.

Определим на множестве C следующие функции:
1) $Ink: C \to C: c = (z_1, \ldots, z_n) \Rightarrow Ink(c) = (z_1, \ldots, z_n + 1)$.
2) $Dec: C \to C: c = (z_1, \ldots, z_n) \Rightarrow Dec(c) = (z_1, \ldots, z_n - 1)$.
3) $Push: C \to C: c = (z_1, \ldots, z_n) \Rightarrow Push(c) = (z_1, \ldots, z_n, 0)$.
4) $Pop: C \to C: c = (z_1, \ldots, z_n) \Rightarrow Pop(c) = (z_1, \ldots, z_{n-1})$.

Переход $t \in T$, может сработать при некоторой разметке M, если $\exists\, c \in C: \forall\ p_i \in \exists\, (x_i, c_i) \in M(p_i): (c_i, c) \in Q$.

Пусть $t \in T$ может сработать и $F_T(t) = (f_T, \wp_T)$ и при этом $\exists\, (x_i, c_i) \wedge f_T(x_1, \ldots, x_n) = (y_1, \ldots, y_m) \wedge \wp_T(x_1, \ldots, x_n, c_1, \ldots, c_n) = (c_1^t, \ldots, c_m^t)$, тогда срабатывание перехода $t \in T$ на разметке M_1 порождает разметку M_2 по следующему правилу: $\forall\ p_i \in t^I: M_2(p_i) = M_1(p_i) \setminus \{(x_i, c_i)\} \wedge \forall\ p_j \in t^O\, M_2(p_j) = M_1(p_j) \cup \{(y_j, c$. Сработать может любой готовый к срабатыванию переход.

Далее, если не указано иного, то функция $\wp_T$ имеет вид $\wp_T(X_1, \ldots, x_n, c_1, \ldots, c_m) = c_1$. И если не указано иного, то функция f_T имеет вид $f_T(x_1, \ldots, x_n) = x_1$.

Моделирование стандартных вершин

Каждой из стандартных вершин, таких как вершины с арифметическими и логическими функциями, а также с другими стандартными функциями языка Cloud Sisal [2], ставится в соответствие один переход, помеченный соответствующей функцией f_T и функцией $\wp_T$ по умолчанию. Количество упорядоченных входных мест равно 2 для бинарных функций, 1 для унарных, и так далее. Количество упорядоченных выходных мест равно 1, если только функция f_T не является многозначной.

Моделирование простых выражений

Простые выражения - это арифметические и логические операции, а также операции с массивами, такие как создание, доступ по индексу или конкатенация. Для каждой такой операции в спецификации графа IR существует стандартная вершина. Простые выражения моделируются с помощью CPN* тривиально. Например, выражение суммы двух элементов моделируется с помощью единственного перехода $t \in T$, множества t^{I}, состоящего из двух мест, множества t^{o}, состоящего из одного места, функции $f_T(x_1, x_2) = x_1 + x_2$, а также функции $\wp_T(x_1, x_2, c_1, c_2) = c_1$, или для краткости, $\wp_T = c_1$. Далее такую функцию $\wp_T$ будет называть тривиальной.

Рассмотрим пример простого многозначного выражения в теле функции на языке Cloud Sisal:

function Main(A, B, C, D : integer returns integer)
A, 1, 1, C + D
end function

На Рис. 1 представлена соответствующая сеть Петри специального вида. Переходы с меткой *id* обозначают $f_T(x) = x$, а также $\wp_T = c_1$.

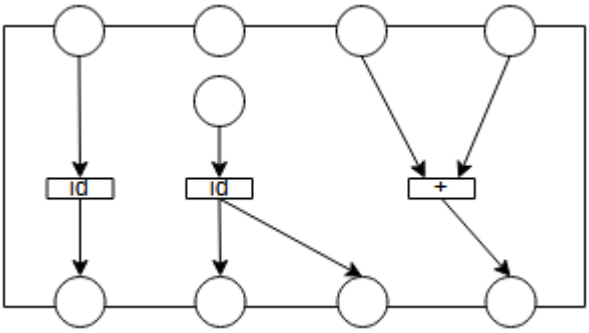

Рис. 1. Пример CPN* для простого выражения

Моделирование условных выражений

Рассмотрим пример условного выражения на языке Cloud Sisal с двумя альтернативами:

if A > B then A+B
else C+D
end if.

Граф внутреннего представления IR для условных вершин не имеет дуг, и семантика задаётся порядком составляющих вершин. Первая вершина имеет столько же входных портов, сколько вершина условного выражения, а количество выходных портов на единицу меньше, чем число альтернатив. В примере выше вершина проверки условия будет иметь четыре входных порта и один выходной. Семантика условного выражения такова, что для вычисления выбирается та альтернатива, номер которой совпадает с номером того выходного порта вершины проверки условия, который имеет наименьший порядковый номер и логическое значение истины. Если все порты вершины проверки условия получили значение «ложь», то выбирается альтернатива, соответствующая выражению else. Альтернатив может быть несколько.

Покажем, что условные выражения языка Cloud Sisal моделируются с помощью CPN* также как простые выражения. В самом деле, спецификация языка Cloud Sisal гарантирует завершение вычисления, при этом возможно специальное ошибочное значение в качестве результата. Следовательно, можно воспользоваться стандартными вершинами создания массива и доступа по индексу, при этом вычислив значения для всех альтернатив.

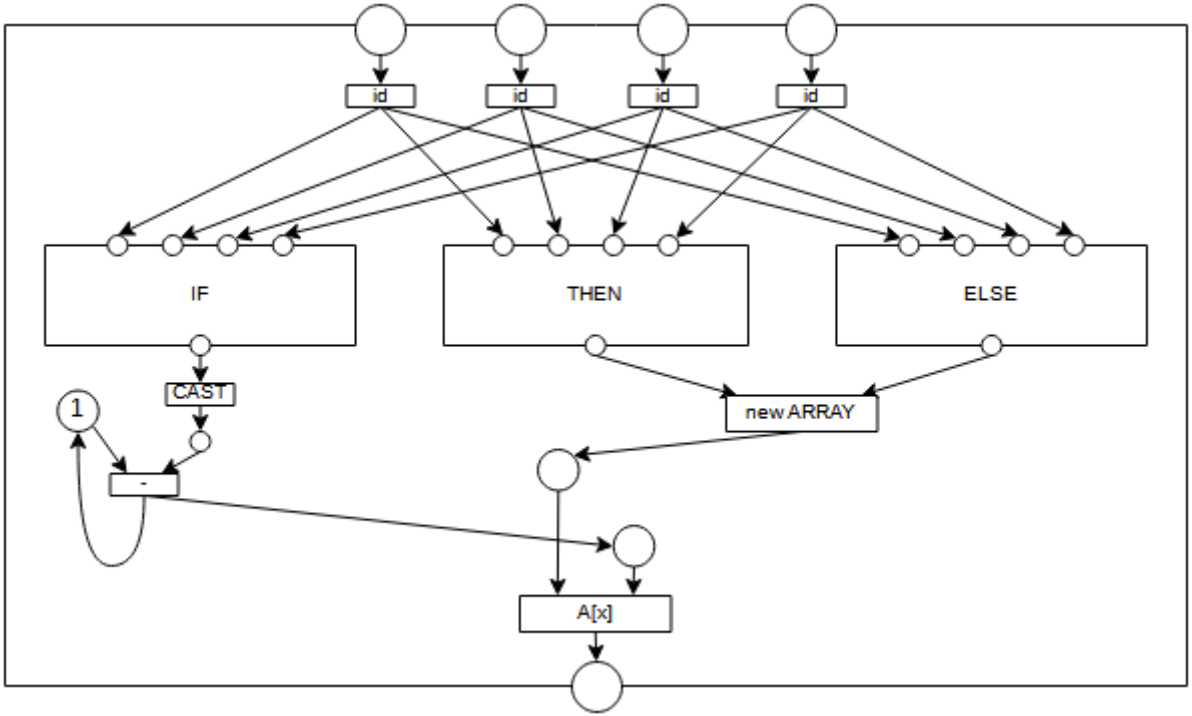

Рис. 2. Пример CPN* для условного выражения с двумя альтернативами

На Рис. 2 представлен пример CPN* для условного выражения. Моделирование условных выражений производится по следующим правилам. Для каждой альтернативы предоставляется копия входных аргументов с помощью использования переходов с меткой id. Далее из выходных значений альтернатив строится массив с помощью стандартной вершины newArray. Из выходных значений вершины вычисления условия строится индекс, по которому следует взять значение из массива с результатами альтернатив. И наконец полученный индекс применяется с помощью стандартной вершины A[x]. Приведённый подход применим к условным выражениями с любым количеством альтернатив, так как согласно спецификации языка Cloud Sisal стандартная вершина newArray может принимать любое конечное количество входных элементов, а также в силу всюду завершающихся вычислений для Cloud Sisal.

Моделирование циклических выражений с независимыми итерациями

Рассмотрим пример циклического выражения с независимыми итерациями на языке Cloud Sisal:

```
for j in 1..M
   repeat
      A := j + M
   returns
      sum of A + j + M
end for
```

На Рис. 3 представлен граф внутреннего представления IR для циклических вершин. Граф IR циклических выражений не имеет дуг, и семантика задаётся порядком следования составляющих вершин. Для таких выражений граф IR содержит три вершины: вершину генератора диапазона индексов, вершину тела цикла, порождающая последовательность значений согласно последовательности индексов, и вершину блока возврата, порождающую последовательность значений

согласно заданным редукциям. Существуют следующие стандартные редукции: сумма всех значений, произведение всех значений, максимальное значение, минимальное значение, последнее по порядку значение и агрегация значений последовательности в массив.

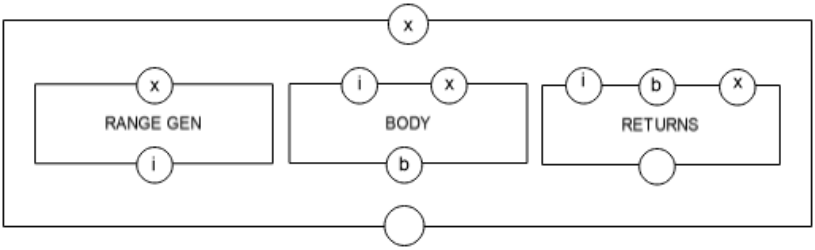

Рис. 3. Граф внутреннего представления для циклического выражения с независимыми итерациями

Вершина генератора диапазона индексов получает те же значения, что и циклическое выражение. Вершина тела цикла получает те же значения, что и циклическое выражение, с добавлением индекса итерации. Вершина возврата результата получает те же значения, что и вершина тела цикла, с добавлением результата итерации тела цикла.

На Рис. 4 представлен пример CPN* для циклического выражения с независимыми итерациями. Моделирование условных выражений производится по следующим правилам. Для вершин генератора диапазона, тела цикла и возврата результатов предоставляется копия входных аргументов с помощью использования переходов с меткой id. Вершина RangeGen порождает конечный набор индексных значений, упорядоченных относительно цвета. Если вершина RangeGen имеет только один выходной порт, то цвета индексных значений нужны только для редуцирования последовательности значений в заданном порядке для данного типа вершин. Если же вершина RangeGen имеет больше одного выходного порта, то цвета индексных значений дополнительно гарантируют корректность входных данных для вершин тела цикла и возврата результата.

Переход с меткой CO помечен функциями $f_T(x_1, x_2) = (x_2, x_2)$ и $\wp_T(x_1, x_2, c_1, c_2) = (c_1, c_2)$ и предназначен для генерации копий значений, являющихся входными для циклического выражения, и имеющих цвета, соответствующие цветам индексных значений. Для каждой редукции добавлены место с меткой 0*, содержащее нейтральное значение для заданной редукции, переход с меткой id*, имеющий $f_T(x_1, x_2) = (x_2, x_2)$ и $\wp_T(x_1, x_2, c_1, c_2) = (c_1+1, c_2)$, а также переход, помеченный меткой +*, имеющий $f_T(x_1, x_2) = x_1+x_2$ и $\wp_T(x_1, x_2, c_1, c_2) = c_1+1$. Перечисленные место и два перехода обеспечивают функциональность заданной редукции для последовательности значений, заданных в порядке, соответствующем цветам индексных значений.

Для циклических выражений с зависимыми итерациями, где в вершину тела цикла дополнительно передаётся результат предыдущей итерации (циклические переменные), также потребуется дополнительный переход с функцией $\wp_T(x_1, x_2, c_1, c_2) = c_1+1$, чтобы гарантировать возможность вычисления следующей итерации. Кроме того потребуется дополнительная вершина для проверки выхода из цикла посредством использования цветов из множества $\widetilde{C}$, а также отдельная вершина, предоставляющая начальные значения для циклических переменных.

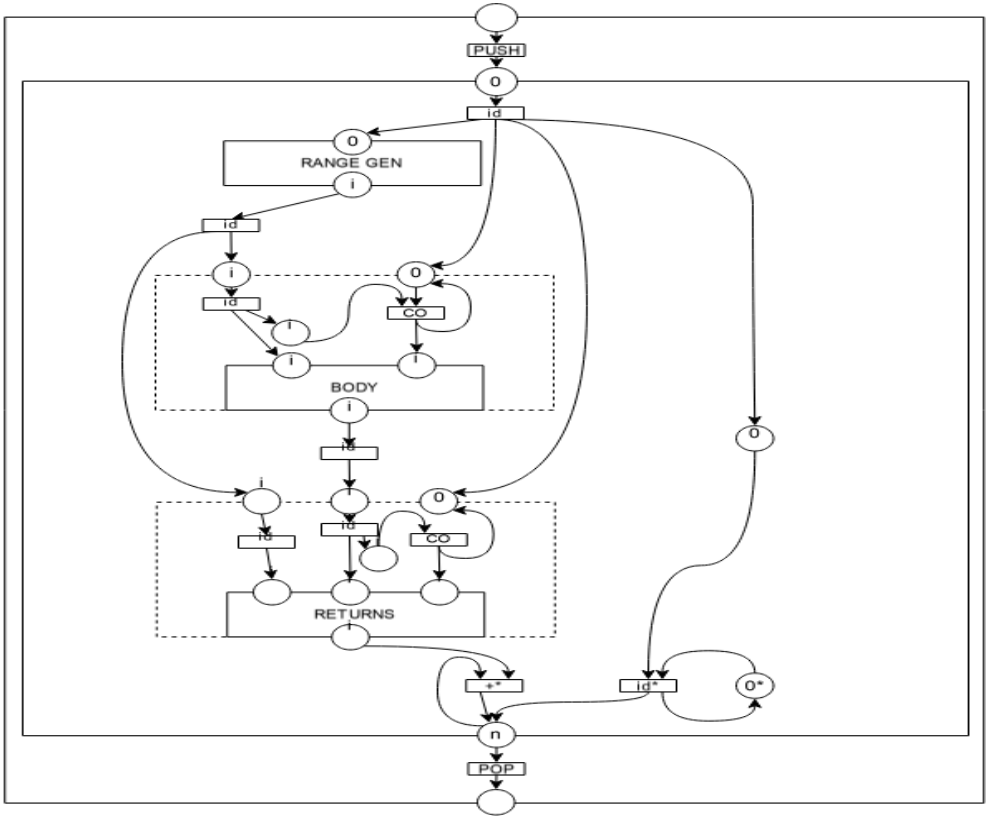

Рис. 4. Пример CPN* для циклического выражения с независимыми итерациями

Моделирование рекурсивных функций

Рассмотрим пример рекурсивной функции на языке Cloud Sisal:

```
function Factorial(n: integer returns integer)
    if n <= 1 then 1
            else n * Factorial(n - 1)
    end if
end function
```

На Рис. 5 представлен граф внутреннего представления IR для такой рекурсивной функции, который задаёт семантику вычислений неявным образом. Как и для условных выражений, вычисления начинаются с вершины условия. Первый по порядку выходной порт, получивший логическое значение истины, соответствует той альтернативе, которая должна быть использована для вычисления результирующих значений всего выражения. Нумерация альтернатив начинается с единицы. Выходных портов вершины условия на 1 меньше, чем альтернатив. Таким образом, если все выходные порты вершины условия получают логическое «ложь», то выбирается последняя альтернатива, соответствующая ветке else. В представленном на Рис.3 графе IR для рекурсивной функции присутствует стандартная вершина CALL, которая обозначает вызов функции. Количество входных портов для вершины CALL на один больше, чем количество входных портов вызываемой функции. В первый порт вершины CALL передаётся ссылка на вызываемую функцию.

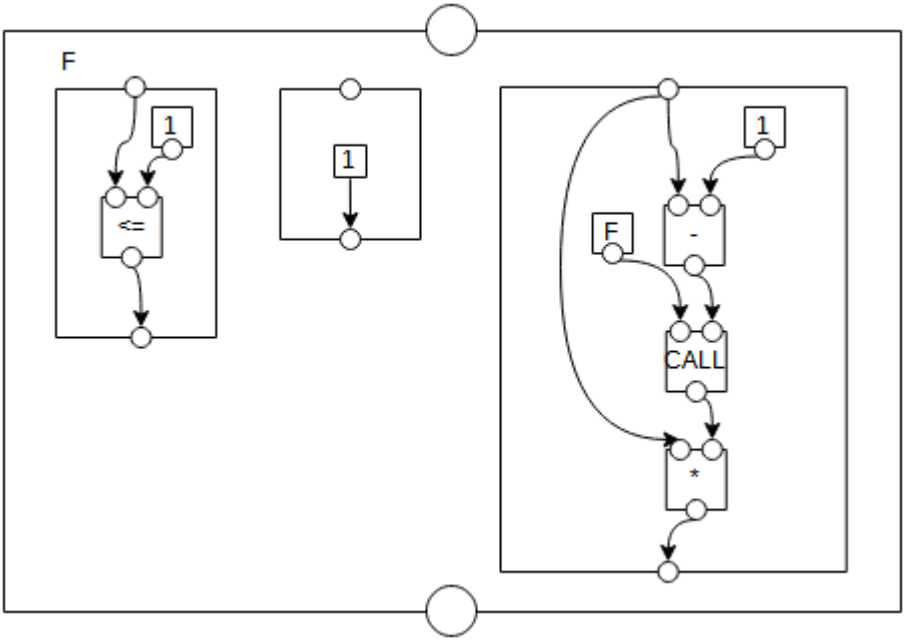

Рис. 5. Пример графа внутреннего представления для рекурсивных функций

На Рис. 6 представлена схема построения CPN* для рекурсивных функций. Моделирование рекурсивных функций производится по следующим правилам. Вершине условия соответствует переход с меткой IF для краткости. Переходы с меткой C++ имеют функцию $\wp_T(x_1, c_1) = c_1 + 1$ или $\wp_T(x_1, c_1) = (c_1 + 1, c_1 + 1)$, в зависимости от количества мест в множестве t°. Так как в условном выражении рекурсивной функции только две альтернативы, то переход условного выражения в CPN* имеет ровно одно выходное место, которое получает фишку со значением «истина», если следует вычислять базовую альтернативу, или «ложь», если следует использовать альтернативу с рекурсивным вызовом. Переход с меткой CAST имеет функцию $f_T(x_1) = (1 : x_1 = true; 2 : x_1 = false)$. Переход с меткой C+=X увеличивает цвет значения фишки на значение из входного места, то есть на 1, если следует вычислять

альтернативу без рекурсивного вызова, и на 2, если следует вычислять альтернативу с рекурсивным вызовом. Или другими словами, он имеет функцию $\wp_T(x_1, c_1) = c_1 + x_1$. Так как фишки одного и того же цвета будут предоставлены только для одной альтернативы, то вычисления начнутся только для неё. Причём, если активируется переход альтернативы с рекурсивным вызовом, то после подготовки значений для «вызова» рекурсивной функции цвет фишек будет изменён с помощью перехода с меткой PUSH, и фишки будут переданы в места соответствующие входным портам рекурсивной функции. Аналогично, при «возврате» фишек из рекурсивного вызова цвет фишек будет изменён с помощью перехода с меткой POP для приведения цвета к «оригинальному». Кроме того, добавлено специальное место для хранения копий фишек с цветом, установленным перед реализацией рекурсивного вызова. Данные копии фишек предназначены для реализации «возврата» из рекурсивного вызова.

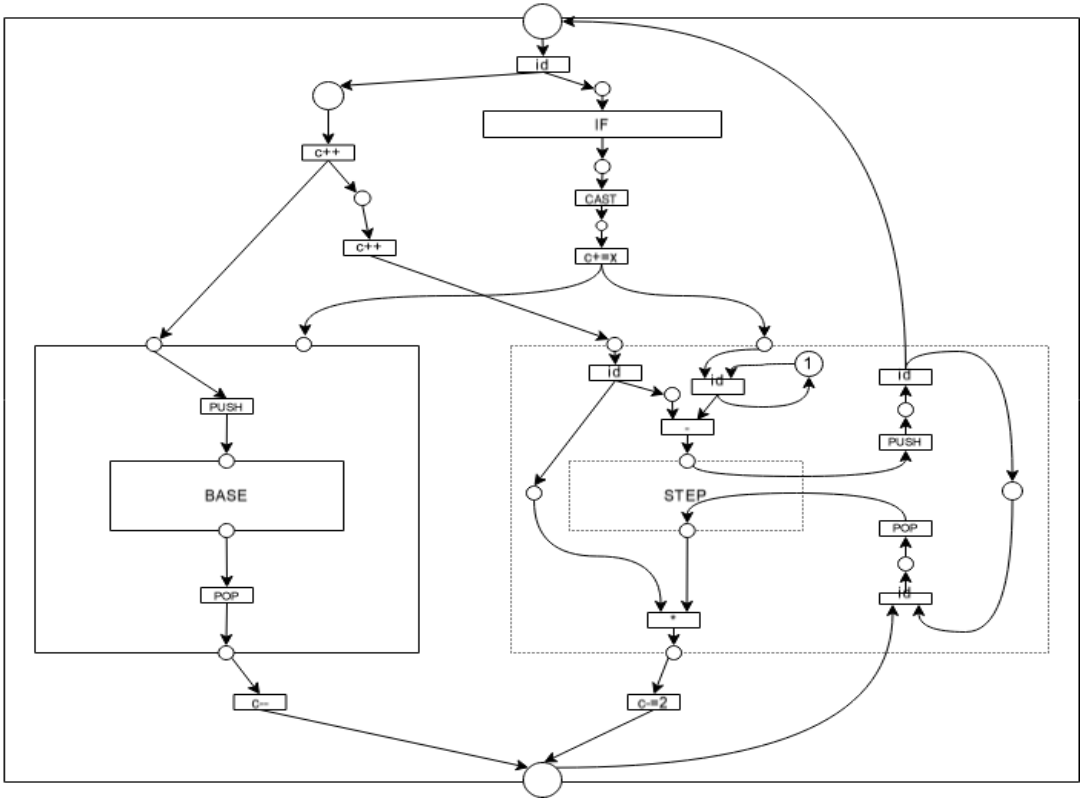

Рис. 6. Пример CPN* для рекурсивных функций

Далее покажем корректность построенной модели с помощью нескольких утверждений.

Утверждение 1. Если существует цвет такой, что в каждом входном месте CPN* существует фишка этого цвета, то по завершении вычислений выходные места CPN* содержат фишки того же цвета.

Доказательство. Докажем это утверждение индукцией по глубине вложенности CPN*. База индукции представляет собой CPN*, построенные по простым вершинам IR, таким как вершины арифметических и логических операций, а также операций над массивами, включая создание или конкатенацию массивов. Таким вершинам по построению соответствует один переход с функцией $\wp_T$ имеющей вид $\wp_T(x_1, \ldots, x_n, c_1, \ldots, c_m) = c_1$. Соответственно, если CPN* состоит из одного такого перехода, и входные места получают фишки одинаковых цветов, то выходное место также получает фишку того же цвета.

Теперь рассмотрим CPN* с глубиной вложенности n, и пусть для всех CPN* с глубиной вложенности $n-1$ и меньше утверждение выполняется. И рассмотрим возможные варианты CPN*, соответствующие выражениям языка Cloud Sisal. Арифметическим или логическим выражениям соответствует связный граф внутреннего представления IR, где внутренними вершинами являются либо простые вершины арифметических или логических операций, либо составные вершины с уровнем вложенности меньше n. Таким образом, для соответствующей CPN* следует, что цвета фишек в выходных местах совпадают с цветами во входных местах, если фишек достаточно для начала срабатывания переходов. Доказательство для CPN*, соответствующей условному выражению, повторяет доказательство для арифметических или логических выражений.

Рассмотрим CPN*, соответствующие циклическим выражениям с независимыми итерациями языка Cloud Sisal. По построению каждый переход в такой CPN* является либо переходом типа id, который передаёт фишку без изменения

цвета и значения, либо переходом с CPN* с уровнем вложенности меньше n, либо переходом с функцией f_T, соответствующей одной из функций редукции цикла. В примере на Рис. 4. (цикл ForAll) это переходы с функцией сложения и умножения. Причём начальное значение редукции сложения вычисляется с помощью перехода с функцией $f_T(x_1, x_2) = (x_1 * x_2, x_2)$, где в первое по порядку входное место попадает фишка со значением из входного места CPN* , а во второе по порядку входное место попадает фишка со значением 0 универсального цвета. Но, так как $\wp_T$ перехода имеет вид $\wp_T(x_1, x_2, c_1, c_2) = (c_1, c_2)$, то для этого перехода цвет фишки с выходным значением для инициализации редукции равнен цвету фишки полученной данной CPN* в одном из входных мест. Для переходов с CPN* с уровнем вложенности меньше n сохранение цвета выполняется по индукционному предположению.

Рассмотрим CPN*, соответствующие рекурсивным функциям языка Cloud Sisal. Докажем утверждение для таких CPN* индукцией по числу рекурсивных вызовов k. Для $k = 0$ рекурсивных вызовов нет, и по построению CPN* будет использована альтернатива с меткой BASE, при этом альтернатива STEP не получит фишек с одинаковыми цветами для начала вычислений. По построению, при вычислениях по альтернативе с меткой BASE фишки проходят через последовательность переходов с метками C++, PUSH, BASE, POP, C--. Соответственно, если входные места CPN* получили фишки с цветом $c = (z_1, \ldots, z_m)$, то переход с вложенной CPN* получит во входные места фишки с цветом $c^1 = (z_1, \ldots, z_m + 1, 0)$ и, по индукционному предположению, вернёт фишки с таким же цветом. Далее последовательность переходов POP, C-- преобразует цвет фишек к оригинальному цвету $c^1 = (z_1, \ldots, z_m)$. И так как других готовых срабатыванию переходов в CPN* нет, то для базы индукции без рекурсивных вызовов утверждение доказано. Рассмотрим также базу индукции для $k = 1$, что

соответствует использованию альтернативы с меткой STEP, при этом альтернатива BASE не получит фишек с одинаковыми цветами во входных местах. По построению, если входные места CPN* получили фишки с цветом $c=(z_1,\ldots,z_m)$, то входные места альтернативы STEP получают фишки с цветом $c^1=(z_1,\ldots,z_m+2)$. Далее, после подготовки значений фишек к рекурсивному вызову, срабатывает переход с меткой PUSH, который отправляет фишки с цветом $c^2=(z_1,\ldots,z_m+2,0)$ во входные места CPN*, а также в специальное место с меткой S. Так как для фишек с цветом $c^2=(z_1,\ldots,z_m+2,0)$ $k=0$, то выходные места CPN* получают фишки с тем же цветом c^2. И так как место с меткой S также содержит фишку с цветом c^2, срабатывает переход с меткой POP альтернативы STEP. Далее происходит обработка результатов рекурсивного вызова, и в выходные места CPN* передаются фишки с цветом $c=(z_1,\ldots,z_m)$. Так как место с меткой S не содержит фишек с цветом $c=(z_1,\ldots,z_m)$, вычисления CPN* останавливаются с фишками цвета $c=(z_1,\ldots,z_m)$ в выходных местах, что и требовалось доказать для $k=1$.

Рассмотрим случай с $k>1$. Тогда, если во входные места CPN* подаются фишки с цветом $c=(z_1,\ldots,z_m)$, то после срабатывания перехода с меткой PUSH во входные места CPN* попадают фишки с цветом $c^1=(z_1,\ldots,z_m+2,0)$ и значениями, для которых число рекурсивных вызовов будет $k-1$. Следовательно, по индукционному предположению в выходные места CPN* будут переданы фишки с тем же цветом c^1, и так как место с меткой S также содержит фишку с цветом c^1, то в выходные места CPN* будут переданы фишки с цветом $c=(z_1,\ldots,z_m)$ и при этом место с меткой S будет пустым. Следовательно, вычисления останавливаются, и утверждение доказано для случая $k>1$.

Таким образом цвет фишек в выходных местах CPN* совпадает с цветом фишек, переданных во входные места, если фишек было достаточно для запуска вычислений.

Утверждение 2. Если существует цвет такой, что в каждом входном месте CPN* существует фишка этого цвета, то по завершении вычислений выходные места CPN* содержат фишки с корректными выходными значениями.

Доказательство. Докажем это утверждение индукцией по глубине вложенности CPN*. База индукции представляет собой CPN*, построенные по простым вершинам IR, таким как вершины арифметических и логических операций, а также операций над массивами, и включая создание или конкатенацию массивов. Таким вершинам по построению соответствует один переход с функцией f_T, корректной по определению. Соответственно, если CPN* состоит из одного такого перехода, то при срабатывании выходное место получает фишку с корректным значением.

Теперь рассмотрим CPN* с глубиной вложенности n, и пусть для всех CPN* с глубиной вложенности $n-1$ и меньше утверждение выполняется. И рассмотрим возможные варианты CPN*, соответствующие выражениям языка Cloud Sisal. Арифметическим или логическим выражениям соответствует связный граф внутреннего представления IR, где внутренними вершинами являются либо простые вершины арифметических или логических операций, либо составные вершины с уровнем вложенности меньше n. Таким образом, для соответствующей CPN* следует, что, если фишек достаточно для начала срабатывания переходов, то промежуточные переходы обеспечивают фишки с корректными значениями в выходных местах CPN* для всего выражения. Доказательство для CPN*, соответствующей условному выражению повторяет доказательство для арифметических или логических выражений.

Рассмотрим CPN*, соответствующие циклическим выражениям с независимыми итерациями языка Cloud Sisal. По построению каждый переход в такой CPN* является либо

переходом типа id, который передаёт фишку без изменения цвета и значения, либо переходом с CPN* с уровнем вложенности меньше n, либо переходом с функцией f_T, соответствующей одной из функций редукции цикла. По индукционному предположению каждый такой переход передаёт в свои выходные места корректные значения. Таким образом, выходные места перехода блока редукций, помеченного меткой returns, получают последовательность фишек с корректными значениями. Так как вершина с меткой n также получает фишку с нейтральным значением, являющимся корректным для функции f_T, то переход функции редукции срабатывает до тех пор, пока в выходном месте перехода блока редукций не останется ни одной фишки. Соответственно в месте с меткой n останется только фишка со значением редуцированной последовательности.

Рассмотрим CPN*, соответствующие рекурсивным функциям языка Cloud Sisal. Докажем утверждение для таких CPN* индукцией по числу рекурсивных вызовов k. Для $k=0$ рекурсивных вызовов нет, и по построению CPN* будет использована альтернатива с меткой BASE, при этом альтернатива STEP не получит фишек с одинаковыми цветами для начала вычислений. По построению, при вычислениях по альтернативе с меткой BASE фишки проходят через последовательность переходов с метками C++, PUSH, BASE, POP, C--. Соответственно, если входные места CPN* получили фишки с корректными значениями, то и переход с вложенной CPN* получит во входные места фишки с корректными значениями, и по индукционному предположению, вернёт фишки также с корректными значениями. Далее последовательность переходов POP, C-- сохраняет значения фишек. И так как других готовых срабатыванию переходов в CPN* нет, то для базы индукции без рекурсивных вызовов утверждение доказано. Рассмотрим также базу индукции для $k=1$, что соответствует использованию альтернативы с меткой STEP, при этом альтернатива BASE не получит фишек с одинаковыми цветами во входных местах. По построению, если

входные места CPN* получили фишки с корректными значениями, то и входные места альтернативы STEP получают фишки корректными значениями. Далее, после подготовки значений фишек к рекурсивному вызову, срабатывает переход с меткой PUSH, который отправляет фишки с корректными значениями во входные места CPN*, а также в специальное место с меткой S. По индукционному предположению выходные места CPN* получают фишки с корректными значениями Далее происходит обработка результатов рекурсивного вызова, и в выходные места CPN* передаются фишки с корректными значениями. Так как место с меткой S не содержит фишек, вычисления CPN* останавливаются с фишками в выходных местах, которые содержат корректные значения. что и требовалось доказать для $k=1$.

Рассмотрим случай с $k>1$. Тогда, если во входные места CPN* подаются фишки с корректными значениями, то после срабатывания перехода с меткой PUSH во входные места CPN* попадают фишки с корректными значениями, для которых число рекурсивных вызовов будет $k-1$. Следовательно, по индукционному предположению в выходные места CPN* будут переданы фишки с корректными значениями, и так как место с меткой S также содержит фишку, то в выходные места CPN* будут переданы фишки с корректными значениями, и при этом место с меткой S будет пустым. Следовательно, вычисления останавливаются, и утверждение доказано для случая $k>1$.

Таким образом, в выходных местах CPN* содержатся фишки с корректными значениями.

Утверждение 3. Если существует n различных цветов, таких что для каждого из них во всех входных местах CPN* содержатся фишки такого цвета, то по завершении вычислений каждое выходное место CPN* содержит n фишек тех же цветов, и их выходные значения корректны для каждого цвета.

Доказательство. По определению для активации перехода требуется существование фишек сравнимого цвета во всех входных местах CPN*. Если таких цветов больше одного, то, по

определению срабатывания активного перехода, из входных мест убираются фишки только одного из цветов. Соответственно переход остаётся активным по другому цвету. Если все n цветов различны, то переход CPN* может сработать по каждому из них в произвольной последовательности, и по Утверждению 2 значения в выходных местах будут корректными для каждого из цветов. Что и требовалось доказать.

Определение. Добавлением или установкой точки останова для перехода t будем называть добавление места без фишек в множество t^I.

Теорема 1. Если существует n различных цветов, таких что для каждого из них во всех входных местах CPN* содержатся фишки такого цвета, то для любой расстановки точек останова в CPN* для переходов $t_1, \ldots, t_n$ значения фишек в местах t_i^I будут корректны для любого $i \in (1, \ldots, n)$.

Доказательство. В силу Утверждения 3, не теряя общности, достаточно рассмотреть случай n = 1. Если CPN* не содержит переходов, соответствующих IR-вершинам циклов или рекурсивных функций, то фишки всегда имеют один и тот же цвет. В этом случае корректность значений в местах из множества t_i^I обеспечивается Утверждением 2, так как любой переход в CPN* представим в виде CPN*. Если же в CPN* существуют переходы, соответствующие IR-вершинам циклов или рекурсивных функций, то в процессе вычисления могут возникать фишки разных цветов. При этом переходы, принадлежащие CPN*, соответствующим переходам с метками Body и Returns для циклов и переходам с метками Base и Step для рекурсивных функций, могут получать в свои входные места фишки со значениями, соответствующими разным вычислениям. Однако по построению CPN* цвета таких фишек будут различными. Следовательно, по Утверждению 3 значения фишек всегда будут корректными, что и требовалось доказать.

Теорема 1 доказывает корректность моделирования IR представления Cloud Sisal программ и обеспечивает корректность визуальной отладки на графах внутреннего представления IR с помощью механизма точек останова. При реализации визуальной отладки также допустимо изменить значения фишек во входных местах деактивированных переходов и активировать переходы для продолжения дальнейших вычислений, что также соответствует общепринятой функциональности точек останова.

Заключение

В данной работе представлен метод моделирования Cloud Sisal программ с помощью CPN*, позволяющий явным образом реализовать семантику языка для всех видов выражений, с явной семантикой (простые выражения) и неявной семантикой (условные, циклические и рекурсивные выражения), а также показана корректность данного метода. Целью разработки данного метода является обеспечение возможности визуальной отладки Cloud Sisal программ как продолжение цикла работ по визуализации Cloud Sisal программ.

По построению для каждой вершины графа IR существует соответствующий переход t в CPN*, причём входные и выходные порты вершины соответствуют множествам t^I и t^o. Таким образом, приписывая точку останова к некоторому порту или вершине графа IR, можно также связать эту точку останова с соответствующими местами или переходами в CPN*. Соответственно, активация такого механизма отладки как точка останова для порта или вершине графа IR, позволяет внести изменение в CPN* в процессе срабатывания переходов. Например, эффекта точки останова можно добиться добавлением входного места без фишек для выбранного перехода. И, соответственно, значения в местах из множества t^I можно изменить без влияния на вычисления, на которые не повлияло добавление места без фишек, что соответствует известной функциональности точек останова в популярных IDE.

Альтернативным вариантом активации точек останова является изменение цвета фишек в заданном месте на цвета из множества $\widetilde{C}$. Данный вариант активации точек останова удобен тем, что не вызывает конфликтов с арностью функций f_T и $\wp_T$ в отличие от варианта с дополнительным местом без фишек.

Данный метод имеет преимущество при практической реализации рекурсивных функций по сравнению с реализацией с помощью простых сетей Петри в рамках CSIRI [8], где применяется копирование сети Петри для реализации очередного рекурсивного вызова. Использование иерархии цветов CPN* позволяет реализовать рекурсивные вызовы с помощью единственного экземпляра сети Петри, что является преимуществом данного метода.

Список литературы

1. Касьянов В.Н., Касьянова Е.В., Методы и система облачного параллельного программирования, материалы XIV Международной Азиатской школы-семинара «Проблемы оптимизации сложных систем», Алматы, 2018, сс. 298-307.
2. Касьянов, В.Н. Методы и система облачного параллельного программирования / В.Н. Касьянов, Е.В. Касьянова // Информатика: проблемы, методология, технология: Сборник материалов XIX международной научно-методической конференции. Под ред. Д.Н. Борисова – Воронеж: Вэлборн, 2019. – С. 1552-1556.
3. Касьянов В.Н., Касьянова Е.В. , Язык программирования Cloud Sisal, – Новосибирск, 2018. – 45 с. – (Препринт/РАН, Сиб. отд-ние, ИСИ; N181).
4. Касьянов В.Н., Евстигнеев В.А., Графы в программировании: обработка, визуализация и применение, БХВ-Петербург, Санкт-Петербург, 2003.
5. Гордеев, Д.С. Визуализация внутреннего представления программ на языке Cloud Sisal // Научная визуализация. – 2016. – Т. 8. – №2. – С. 98-106.

6. Котов В.Е. Сети Петри. – М.: Наука. Главная редакция физико-математической литературы, 1984. – 160 с.
7. Гордеев Д.С., Обзор техник визуализации алгоритмов на графах, Научная визуализация (2018), Том 10, 18-48, doi:10.26583/sv.10.1.02.
8. Касьянов В.Н., Гордеев Д.С., Программа интерпретации внутреннего представления Cloud Sisal программ (CSIRI). – Свидетельство о государственной регистрации программы для ЭВМ RU 2022613108. – 2022.

Утверждено к печати в электронном виде
Редакционным советом
Института систем информатики СО РАН

Гордеев Д.С.

Модель визуальной отладки Cloud Sisal программ с помощью цветных сетей Петри

Препринт

Редактор
Рецензент