УДК 006.72

А.А. Воевода, Д.О. Романников

АЛГОРИТМ ОБЪЕДИНЕНИЯ ЧАСТЕЙ ОРИЕНТИРОВАННОГО ГРАФА

Воевода А.А., Романников Д.О. **Алгоритм объединения частей ориентированного графа.**

Аннотация. Рассматривается задача объединения графов с общей частью, которые были получены в результате серии моделирований сети Петри с использованием программного пакета Colored Petri Nets Tools, в котором адресное пространство процесса ограничено 2^{32} байтами, начиная с различных вершин и при различных начальных условиях. Для ее решения необходимо определить общую часть графов, выполнить разрез таким образом, чтобы их общая часть осталась только в одном из начальных графов, и составить таблицу соответствия (переходов) между вершинами графов для возможности осуществления переходов между ними. Изначально предполагается, что графы представлены в виде списков смежности, но в процессе работы алгоритма они преобразовываются в хеш-таблицы для быстрого определения общей части графов, которое реализуется при помощи обхода одного из графов и проверки наличия вершин во втором. Составление таблицы переходов между графами осуществляется при помощи обхода графа по парам «родительская-дочерняя» вершины, в ходе которой проверяются условия добавления узлов в таблицу переходов. Предлагается алгоритм решения задачи объединения частей ориентированного графа и приведен пример его использования.

Ключевые слова: графы, программное обеспечение, алгоритмы, объединение графов, разрез графа, разрезающее множество.

Voevoda A.A., Romannikov D.O. **Algorithm of Uniting of Parts of Oriented Graph.**

Abstract. The task of uniting graphs with a common part that were received as the result of series of simulations of a Petri net with using of program package Colored Petri Nets Tools in which a process address space is restricted by 2^{32} bytes starting with different vertices with different initial conditions is considered. For its solving it is necessary to determine the graphs common part, to perform graphs cutting in such a way that their common part will be only one of the initial graphs, and compose a table of accordance (transitions) between the graphs vertices for possibility of making the transitions between them. Firstly, we assume that the graphs are represented in form of adjacency lists, but they are converted into hash tables during the algorithm work. It's required for fast determination of the common part of the graphs that are implemented with help of traversing one of the graph and checking that the nodes exist in the second graph. Composing of the transition table is realized with help of graph traversal by "parents-child" vertex pairs and check that one of the nodes of pair can be added to the table. The algorithm for solving the problem of uniting the parts of directed graph is offered, and an example of its use.

Keywords: graphs, software, algorithms, graphs union, cutting graph, cutting set.

1. Введение. В настоящее время отсутствие критических ошибок в пользовательских сценариях программного обеспечения (ПО) на практике решается в основном за счет использования методологических способов [1], которые позволяют найти наиболее простые ошибки в основных сценариях использования ПО в лабораторной обстановке. Формальные инструменты анализа ПО, к которым можно отнести статические анализаторы [2–4], инструменты проверки моделей [5] и

др. [6], позволяют выявить достаточно широкий класс ошибок, включая те, которые трудно обнаружить в лабораторных условиях. Например, в работах [7-9] в качестве формального инструмента верификации используются сети Петри, с помощью которых строится модель ПО, а ее интерпретация позволяет определить некоторые классы ошибок. В приведенных работах для анализа и интерпретации модели сетей Петри используется программный пакет CPN Tools 4.0.1 (Colored Petri Nets Tools). Однако размеры решаемых задач анализа ПО превосходят возможности используемых инструментов: пакет моделирования сетей Петри CPN Tools имеет ограничение для хранения графа пространства состояний (ГПС) в 2^{32} байт из-за использования 32х битной архитектуры ML компилятора. При решении вышеприведенных задач, особенно в случаях многопоточности [7, 8], достаточно часто возникает проблема, что ГПС не помещается в память процесса в ~2 ГБ (2 ГБ для пространства ядра и 2 ГБ для пользовательского пространства). Для обхода этого ограничения можно выполнить серию дополнительных моделирований с места, где ГПС перестал умещаться в адресное пространство. Итогом серии моделирований является набор ГПС, где пары графов имеют общие части. После этого нужно удалить общие части из графов и составить таблицу переходов, дополняющую таблицы переходов начальных графов.

2. Постановка задачи. Из всей серии моделирований рассмотрим случай для двух ориентированных графов (далее просто графов) ($G_i = \{V_i, E_i\}$, где V – множество вершин, а E – множество переходов), т.к. решив данную задачу для случая из двух графов можно применить полученное решения для остальных графов из серии моделирования.

С точки зрения алгоритма не принципиально, из какого графа будет удалена общая часть, поэтому для определенности будем считать, что общая часть останется в графе G_2 и будет удалена из G_1.

После удаления общей части необходимо решить задачу обеспечения переходов между двумя графами при его обходе. Для этого нужно решить задачу построения таблицы переходов вида {«узел»: [«список переходов»], …}, где ключом таблицы является узел графа, а значением – список переходов.

Таким образом, в работе рассматривается задача для двух графов $G_1 = \{V_1, E_1\}$, $G_2 = \{V_2, E_2\}$ с общей частью $V_3 \in V_1$, $V_3 \in V_2$, для которых требуется: 1) удалить общую часть V_3 из графа G_1; 2) построить таблицу переходов между двумя графами.

3. Решение. Рассмотрим рисунок 1, на котором изображены два графа: первый G_1, с вершиной (подразумевается вершина, с которой начиналось моделирование) N_{r1}, представлен в виде белых вершин со

сплошными линиями переходов, второй G_2, с вершиной N_{r2}, представлен в виде закрашенных вершин с пунктирными линиями переходов. Общая часть графов представлена в виде заштрихованных вершин. Размер общей части неизвестен. При этом существуют переходы из G_1 не только в вершину графа G_2, но и в его дочерние узлы. При этом могут возникать ситуации, когда: 1) в G_1 есть переходы в общую часть с G_2 (переход e_1); 2) из G_2 есть переходы в G_1 (внутри общей части графов G_1 и G_2) (переход e_2). Ситуация, где из графа G_2 есть переход в граф G_1 (не в общую часть), невозможна, т.к. при этом узел графа G_1, в который осуществляется переход, должен принадлежать графу G_2.

Предположим, что графы представлены в виде хеш-таблиц, где узлы являются ключами. Тогда определить общую часть графов G_1 и G_2 достаточно просто: потребуется лишь обойти один из графов и проверить наличие его узлов в другом. Можно использовать обход графа в ширину или глубину [10-13]. После этого необходимо удалить общую часть графа из графа G_1.

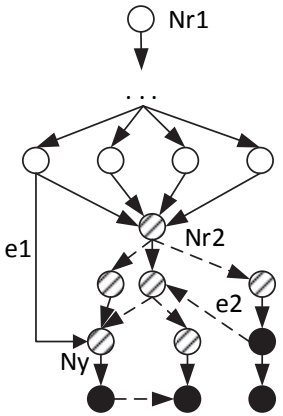

Рис. 1. Два графа с общей частью

Задача составления таблицы переходов между графами схожа с подзадачей определения минимального разреза в задаче о транспортных потоках [14, 15]. При этом таблицу переходов можно составить, найдя разрезающее множество [10] между общей частью и графом G_1. Для определения разрезающего множества необходимо выполнить обход графа G_1 и найти пары, для которых выполняются условия:

1. Существует переход в узел n графа G_2, у которого родительский узел p принадлежит графу G_1, но не принадлежит графу G_2: $e = (p, n), p \in G_1, n \in G_2, p \notin G_2$;

2. Существует переход из узла p принадлежащего общей части графов в узел n, который принадлежит только графу G_1: $e = (p, n), p \in G_1, p \in G_2, n \in G_1, n \notin G_2$.

После этого нужно добавить узел p в таблицу переходов.

4. Алгоритм. Рассмотрим реализацию алгоритма (листинг 1) для решения данной задачи и проиллюстрируем его работу на графах с рисунка 2. Раскраска для обозначения графов аналогична с графами на рисунке 1. Алгоритм приводится в виде псевдокода, в котором, будем предполагать, что переменные передаются по ссылке, т.е. их модификация внутри функции меняет значение в вызывающем коде.

Алгоритм начинает свою работу с функции main, где изначально при помощи функции makeHashMap выполняется предобработка и формируются хеш-таблицы, в которые помещаются узлы графов G_1 и G_2 в качестве ключей, путем обхода графа в ширину.

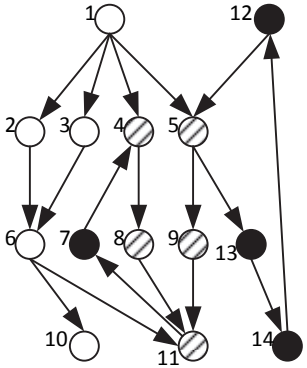

Рис. 2. Пример двух графов с общей «боковой» частью

После этого, согласно вышеприведенному описанию, определяется общая часть графов в функции findCommon (общая часть помещается в переменную graphCommon), в которой выполняется цикл по всем ключам графа G_1 из хеш-таблицы и в цикле выполняется проверка на наличие этих ключей в хеш-таблице графа G_2. Переменная graphCommon будет иметь значение {4: true, 5: true, 8: true, 9: true, 11: true}.

```
makeHashMap(graph, hashMap):
  Queue q // очередь
  q.push (вершина графа graph, с которой будем начинать обход)
  while пока в очереди есть элементы:
    Node n = q.pop()
    hashMap[n] = n
    for каждой вершины child ∈ n.children:
      if вершина child не содержится в хеш-таблице hashMap
        q.push(child)

findCommon (g1GraphHash, g2GraphHash, graphCommon):
  for каждой вершины node из ключей g1GraphHash:
    if вершина node содержится в хеш-таблице g2GraphHash:
      graphCommon[node] = true

findBoundAndRemoveCommon (g1HashMap, g2HashMap, graphCommon, bound):
  Queue q // очередь
  HashMap usedNodes // хеш-таблица для обхода графа
  q.push( пара [nil, g1HashMap.top] с которой будем начинать обход)
  while пока в очереди есть элементы:
    [Node parent, Node n] = q.pop()
    usedNodes[[parent, n]] = true
    if первое условие добавления в таблицу переходов: p ∈ G1, n ∈ G2, p ∉ G2
      bound[n] = переходы из g1HashMap[n] и g2HashMap[n]
    else if второе условие добавления в таблицу переходов: p ∈ G1, p ∈ G2, n
∈ G1, n ∉ G2
      bound[p] = переходы из g1HashMap[p] и g2HashMap[p]
    for каждой вершины child ∈ n.children:
      if пара [n, child] не содержится в хеш-таблице usedNodes:
        q.push([n, child])

  for каждой вершины node из ключей graphCommon:
    if в node из G1 есть переходы отличные от node из G2:
      g2HashMap[node] = переходы g2HashMap[node] и g1HashMap[node]
    g1HashMap.remove(node)


main():
  HashMap g1GraphHash, g2GraphHash, graphCommon, bound
  Graph g1Graph, g2Graph

  makeHashMap(g1Graph, g1GraphHash)
  makeHashMap(g2Graph, g2GraphHash)

  findCommon(g1GraphHash, g2GraphHash, graphCommon)
  findBoundAndRemoveCommon(g1GraphHash, g2GraphHash, graphCommon,
bound)
```

Листинг. 1 Алгоритм объединения частей графов

Для определения таблицы переходов для графов и удаления общей части из графа G_1 используется функция findBoundAndRemoveCommon, в которой выполняется обход графа G_1 по парам «родительский узел - дочерний узел» и каждая пара проверяется на то, является ли переход между этими узлами переходом из разрезающего множества. Дочерние узлы переходов разрезающего множества помещаются в переменную bound. В ходе обхода будут пройдены следующие пары: [nil, 1], [1, 2], [1, 3], [1, 4], [1, 5], [2, 6], [3, 6], [6, 10], [6, 11], [4, 8], [8, 11], [5, 9], [9, 11]. Среди этих пар условие добавления в хеш-таблицу срабатывает для пар [1, 4], [1, 5], [6, 11] и переменная bound будет иметь значение {4: [8], 5: [9, 13], 11: [7]}. Следует заметить, что для узлов вышеприведенной таблицы переходы состоят из переходов обоих графов.

Также в этой функции выполняется удаление общей части графов из G_1. При удалении узлов общей части из графа G_1 возможна такая ситуация, когда существуют два узла принадлежащие обоим графам (узлы 8 и 11 на рисунке 2), а переход между ними есть только в вершине графа, из которого она будет удалена (в данном случае – это граф G_1). Для предотвращения потери переходов при удалении общей части графов необходимо добавить переходы из удаляемых узлов графа G_1 в те же узлы графа G_2 (очевидно, что добавлять имеет смысл только те узлы, которых не было во втором графе).

Перейдем к анализу асимптотической сложности [10–13] алгоритма. В функции makeHashMap выполняется обход графа в ширину, сложность которого определяется как O(n), тогда для последовательных вызовов данной функции для графов G_1 и G_2 асимптотическая сложность алгоритма будет O(n + m), где n – количество узлов в графе G_1, m – в графе G_2.

В функции findCommon также выполняется обход графа G_2, что не меняет общую асимптотическую сложность.

Следующим действием является вызов функции findBoundAndRemoveCommon, в которой определяется разрезающее множество и удаляется общая часть из графа G_1. Вторая часть алгоритма также реализована на обходе графа и не меняет общую сложность. Рассмотрим часть, где выполняется определение общей границы. Данная часть алгоритма является модифицированным вариантом обхода графа в ширину за исключением того, что в обходе участвуют не узлы графа, а пары «родительский узел-дочерний узел», что меняет асимптотическую сложность алгоритма с O(n) на O (n^2). Таким образом, общая асимптотическая сложность алгоритма будет вычисляться

следующим выражением: O (n + m) + O (n) + O (n) + O (n^2), что в итоге приводит к O (n^2). Следует отметить, что вышеприведенная сложность указана для наихудшего варианта, когда все узлы графа соединены между собой. При анализе ПО таких вариантов не возникает и поэтому в решаемых задачах асимптотическая сложность алгоритма близка к O (n+m).

5. Пример. Рассмотрим граф на рисунке 3. Данный граф приводится в работах [9, 16] и использовался для анализа части программного обеспечения работы банкомата. Граф сгенерирован на основании обхода модели в сетях Петри при помощи программного пакета CPN Tools. В узлах графов (на рисунке они представлены в виде прямоугольников с закруглёнными углами) показано номер узла в верхней части, количество входных и выходных переходов в нижних левом и правом частях узла. Предположим, что он был получен по частям: сначала граф G_1, потом граф G_2. Граф G_1 задается соотношением из первой колонки таблицы 1, а граф G_2 соотношением из второй колонки. Покажем работу алгоритма на данном примере.

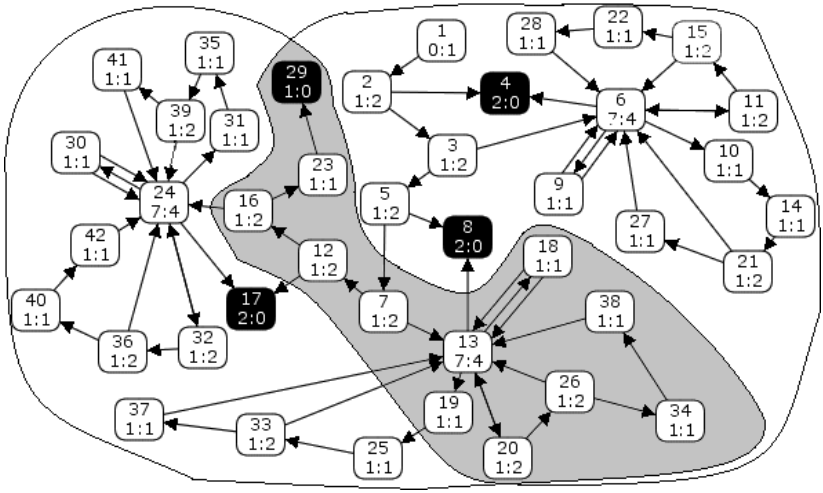

Рис. 3. Часть графа состояний сети Петри, в которой моделируется взаимодействие банкомата и пользователей

Алгоритм начинает свою работу с вызовов функций makeHashMap для обоих графов. Внутри этой функции заполняется переданная хеш-таблица: выполняется обход графа в глубину и каждый узел графа помещается в хеш-таблицу в качестве ключа и ссылка на тот же узел в качестве значения. Для рассматриваемого примера это

имеет место, и алгоритм не завершается на данном этапе. Далее следует вызов функции findCommon, где выполняется перебор всех ключей из хеш-таблицы, соответствующей первому графу, и заполняется переменной graphCommon, если узел принадлежит обоим графам. В данном примере переменная graphCommon будет иметь следующее значение: «{7: true, 12: true, 13: true, 16: true, 18: true, 19: true, 20: true, 23: true, 26: true, 29: true, 34: true, 38: true}». Последним шагом алгоритма является определение границы графов и удаление общей части.

Таблица 1. Представление графов с рисунка 3 в виде списков смежности

Граф G_1		Граф G_2	
[1: [2],	2: [3, 4],	[7: [12, 13],	12: [16, 17],
3: [5, 6],	4: [],	13: [18, 19, 20],	16: [23, 24],
5: [7, 8],	6: [9, 10, 11],	17: [],	18: [13],
7: [12, 13],	8: [],	19: [25],	20: [13, 26],
9: [6],	10: [14],	23: [29],	
11: [15],	12: [16],	24: [17, 30, 31, 32],	25: [33],
13: [8, 18, 19, 20],	14: [21],	26: [13, 34],	29: [],
15: [6, 22],	16: [23],	30: [24],	31: [35],
18: [13],	19: [],	32: [36],	33: [13, 37],
20: [13, 26],	21: [6, 27],	34: [38],	35: [39],
22: [28],	23: [29],	36: [24, 40],	37: [13],
26: [13, 34],	27: [6],	38: [13],	39: [24, 41],
28: [6],	29: [],	40: [42],	41: [24],
34: [38],	38: [13]]	42: [24]]	

При определении границы графов выполняется обход в глубину графа G_1, где для каждой пары [«родительский узел», «дочерний узел»] определяется необходимость его добавления в переменную bound, под которой понимают таблицу переходов между графами. Обход начинается с пары [nil, 1], после которой следует пары [1, 2], [2, 4] и т.д. до пары [5, 7]. Для последней пары выполняется первое условие добавления в переменную bound, которая принимает значение {7: [12, 13]}. Ключ от узла 7 в переменной bound содержит переходы, совпадающие с переходами в том же узле в графе G_1, но если бы из узла 7 был переход в граф G_2 (при этом такой переход содержался бы только в описании графа G_2), то он также бы был добавлен в список. Для пары [13, 8] также сработает условие добавления в переменную bound, которая примет окончательное значение {7: [12, 13], 13: [8, 18, 19, 20]}.

6. Заключение. В работе приведен алгоритм, позволяющий выполнить объединение двух графов с общей частью, с последующим удалением общей части из одного из графов и построение таблицы переходов для однозначных переходов между графами. Применение данного алгоритма позволяет увеличивать размер анализируемого

графа пространства состояния и, тем самым, увеличивать количество состояний в анализируемой программе.

Литература

1. *Орлов С.А.* Технология разработки программного обеспечения // Питер. 2012. 609 с.
2. *Islam S., Krinke J., Binkley D., Harman M.* Coherent clusters in source code // The Journal of Systems and Software. 2014. vol. 88. pp. 1–24.
3. *Burgstaller B., Scholz B., Blieberger J.* A symbolic analysis framework for static analysis of imperative programming languages // The Journal of Systems and Software. 2012. vol. 85. pp. 1418–1439.
4. *Аветисян А. И.* Современные методы статического и динамического анализа программ для автоматизации процессов повышения качества программного обеспечения: дисс. доктора физ. мат. наук // Москва: 2012. 271 с.
5. *Clarke E.M., O. Grumberg, D. Peled.* Model Checking // The MIT Press. 1999. 330 p.
6. *Шудрак М.О., Золотарев В.В.* Модель, алгоритмы и программный комплекс автоматизированного поиска уязвимостей в исполняемом коде // Труды СПИИРАН. 2015. Вып. 42. С. 212–231.
7. *Романников Д.О.* Разработка программного обеспечения с применением UML диаграмм и сетей Петри для систем управления локальным оборудованием дисс. канд. техн. наук // Новосибирск: 2012. 195 с.
8. *Коротиков С.В.* Применение сетей Петри в разработке программного обеспечения центров дистанционного контроля и управления: дисс. канд. техн. наук // Новосибирск: 2007. 216 с.
9. *Марков А.В.* Автоматизация проектированияи анализа программного обеспечения с использованием языка UML и сетей Петри: дисс. канд. техн. наук // Новосибирск: 2015. 176 с.
10. *Cormen T., Leiserson C., Rivest R., Stein C.* Introduction to Algorithms: 3rd Edition // The MIT Press. 2009. 1328 p.
11. *Even S.* Graph Algorithms: 2nd Edition // Cambridge University Press. 2011. 187 p.
12. *Tarjan R.* Data Structures and Network Algorithms // Society for Industrial and Applied Mathematics. 1983.
13. *Sedgewick R.* Algorithms in C++: 3rd Edition // Addison-Wesley Professional. 1998. 752 p.
14. *Goldberg A., Tardos E., Tarjan R.* Network flow algorithms // Springer. 1990. pp. 101–164.
15. *Schhrijver A.* Paths and flows – A historical survey // CWI Quarterly. 1993. pp. 169–183.
16. *Марков, А.В., Воевода А.А.* Анализ сетей Петри при помощи деревьев достижимости // Сб. науч. тр. НГТУ. 2013. №. 71. С. 78–95.

References

1. Orlov S.A. *Tehnologija razrabotki programmnogo obespechenija* [Technology of software development]. Piter. 2012. 609 p. (In Russ.).
2. Islam S., Krinke J., Binkley D. Coherent clusters in source code. *The Journal of Systems and Software*. 2014. vol. 88. pp. 1–24.
3. Burgstaller B., Scholz B., Blieberger J. A symbolic analysis framework for static analysis of imperative programming languages. *The Journal of Systems and Software*. 2012. vol. 85. pp. 1418–1439.
4. Avetisjan A. I. *Sovremennye metody staticheskogo i dinamicheskogo analiza programm dlja avtomatizacii processov povyshenija kachestva programmnogo obespechenija: diss. doktora. fiz. mat. Nauk* [Modern methods of static and dynamic

analysis software for process automation to improve the quality of software: doctor phys. math. thesis]. Moskow: 2012. 271 p. (In Russ.).
5. Clarke E.M., O. Grumberg, D. Peled. Model Checking. The MIT Press. 1999. 330 p.
6. Shudrak M.O., Zolotarev V.V. [Models, algorithms and software for automated vulnerability scan executable code]. *Trudy SPIIRAN – SPIIRAS Proceedings*. 2015. vol. 42. pp. 212–231.
7. Romannikov D.O. *Razrabotka programmnogo obespechenija s primeneniem UML diagramm i setej Petri dlja sistem upravlenija lokal'nym oborudovaniem: diss. kand. tehn. nauk* [Software development using UML diagrams and Petri nets for local control systems equipment Ph.D. thesis]. Novosibirsk: 2012. 195 p. (In Russ.).
8. Korotikov S.V. *Primenenie setej Petri v razrabotke programmnogo obespechenija centrov distancionnogo kontrolja i upravlenija diss. kand. tehn. nauk* [Application of Petri nets in software development centers, remote monitoring and control: Ph.D. thesis]. Novosibirsk: 2015. 216 p. (In Russ.).
9. Markov A. V. *Avtomatizacija proektirovanijai analiza programmnogo obespechenijas ispol'zovaniem jazyka UML i setej Petri: diss. kand. tehn. nauk* [Computer-aided design and analysis software with UML and Petri nets: Ph.D. thesis]. Novosibirsk: 2007. 176 p. (In Russ.).
10. Cormen T., Leiserson C., Rivest R., Stein C. Introduction to Algorithms: 3rd Edition. The MIT Press. 2009. 1328 p.
11. Even S. Graph Algorithms: 2nd Edition. Cambridge University Press. 2011. 187 p.
12. Tarjan R. Data Structures and Network Algorithms. Society for Industrial and Applied Mathematics, 1983.
13. Sedgewick R. Algorithms in C++: 3rd Edition. Addison-Wesley Professional. 1998. 752 p.
14. Goldberg A., Tardos E., Tarjan R. Network flow algorithms. Springer. 1990. pp. 101–164.
15. Schhrijver A. Paths and flows – A historical survey. CWI Quarterly. 1993. pp. 169–183.
16. Markov A.B., Voevoda A.A. [Petri Nets Analysis with help of reachability trees]. *Sbornik nauchnykh trudov NGTU – Collection of scientific papers of NSTU*. 2013. vol. 1 (71). pp. 78–95. (In Russ.).

Романников Дмитрий Олегович — к-т техн. наук, доцент, доцент кафедры автоматики, Новосибирский государственный технический университет. Область научных интересов: верификация, анализ программ. Число научных публикаций — 40. rom2006@gmail.com; пр. Карла Маркса 20, Новосибирск, 630073; р.т.: +7 961 223 8567.

Romannikov Dmitry Olegovich — Ph.D., associate professor, associate professor of automation department, Novosibirsk State Technical University. Research interests: verification, program analysis. The number of publications — 40. rom2006@gmail.com; 20, Karl Marx Avenue, Novosibirsk, 630073; office phone: +7 961 223 8567.

Воевода Александр Александрович — д-р техн. наук, профессор, профессор кафедры автоматики, Новосибирский государственный технический университет. Область научных интересов: полиномиальный синтез, сети Петри, UML диаграммы. Число научных публикаций — 200. voevoda@ucit.ru; пр. Карла Маркса 20, Новосибирск, 630073; р.т.: +79139223092.

Voevoda Alexandr Aleksandrovich — Ph.D., Dr. Sci., professor, professor of automation department, Novosibirsk State Technical University. Research interests: polynomial synthesis, UML diagrams, Petri nets. The number of publications — 200. voevoda@ucit.ru; 20, Karl Marx Avenue, Novosibirsk, 630073; office phone: +79139223092.

РЕФЕРАТ

Воевода А.А., Романников Д.О. **Алгоритм объединения частей ориентированного графа.**

Рассматривается задача объединения частей графа, которые содержат общую частью. Данные части графов могут быть получены различными способами, например, при моделировании сети Петри с использованием программного пакета CPN Tools (в котором есть ограничение адресное пространство процесса в 2^{32} байт из-за использования 32-х битного компилятора языка ML). Для решения задачи объединения частей графа необходимо выполнить следующее: 1) удалить общую часть графов; 2) составить таблицу переходов между графами состоящую из вершин графов для возможности выполнять переходы между частями графов.

В работе предложен алгоритм, в котором предполагается, что изначально графы, представленные в виде списков смежности, преобразуются в хеш-таблицы. Определение общей части графов выполняется с помощью обхода одного из графов и проверки на вхождение его узлов в другой. Составление таблицы переходов между графами осуществляется при помощи обхода графа по парам «родительский - дочерний» узел, в ходе которой проверяется условия добавления узлов в таблицу переходов.

В работе так же оценена асимптотическая сложность алгоритма и представлены примеры его применения. Полученное решение в дальнейшем может быть последовательно применено для объединения всего множества графов.

SUMMARY

Voevoda A.A., Romannikov D.O. **Algorithm of Uniting of Parts of Oriented Graph.**

The problem of uniting parts of the graph, which contains a general part. The given parts of the graphs can be produced by various methods, for example, in a Petri net modeling with using of a software package CPN Tools (which has a limited address space of 2^{32} bytes in cause of using of 32-bit compiler of ML language). To solve the task of uniting parts of the graph it's need to do the following: 1) remove the general part of the graphs; 2) create a table of transitions between graphs consisting of vertices of graphs to be able to perform transitions between parts of the graph.

The algorithm, which assumes that initially presented as graphs adjacency lists are converted into hash tables. Determination of the total of the graph traversal is performed via one of the graphs and check his entry node to another. Making the transition table between the graphs made using graph traversal pairs "parent – child" node in order to verify the conditions of adding nodes to a jump table.

The paper also evaluated the asymptotic complexity of the algorithm and provides examples of its use. The resulting solution can then be applied consistently to unite the whole set of the graphs.