Обход неизвестного графа коллективом автоматов

Игорь Бурдонов, Александр Косачев
{igor, kos}@ispras.ru

Аннотация. Исследование графов автоматами является корневой задачей во многих приложениях. К таким приложениям относятся верификация и тестирование программных и аппаратных систем, а также исследование сетей, в том числе сети интернета и GRID на основе формальных моделей. Модель системы или сети, в конечном счёте, сводится к графу переходов, свойства которого нужно исследовать. За последние годы размер реально используемых систем и сетей и, следовательно, размер их моделей и, следовательно, размер исследуемых графов непрерывно растёт. Проблемы возникают тогда, когда исследование графа одним автоматом (компьютером) либо требует недопустимо большого времени, либо граф не помещается в памяти одного компьютера, либо и то и другое. Поэтому возникает задача параллельного и распределённого исследования графов. Эта задача формализуется как задача исследования графа коллективом автоматов (несколькими параллельно работающими компьютерами с достаточной суммарной памятью). Решению этой задачи посвящена данная работа.

Ключевые слова: исследование графа, обход графа, конечный автомат, расширенный автомат, взаимодействующие автоматы, параллельная обработка, распределенные системы, тестирование.

1. Введение

Задача обхода неизвестного ориентированного графа автоматом используется во многих приложениях. В данной статье подразумевается тестирование детерминированных систем: граф — это граф автомата тестируемой системы, автомат на графе — тестирующая система, а проход по дуге — это тестовое воздействие и наблюдение результата [[1]]. В качестве практического примера можно привести работу [[8]], где выполнялось функциональное тестирование различных подсистем модели процессора: кэш третьего уровня, управление прерываниями и пр. Модельные графы содержали от нескольких тысяч до нескольких миллионов узлов и несколько миллионов дуг. Тест выполнялся максимально на 150 компьютерах.

Автомат-обходчик выполняется на одной машине (процессор с памятью), а наличие нескольких автоматов на разных машинах позволяет существенно

распараллелить работу. При тестировании клонирование тестируемой системы обычно возможно только в начальном состоянии. Поэтому автомат начинает работу с начальной вершины графа. Будем считать, что за один такт создаётся не более одного клона тестируемой системы, то есть не более одного автомата-обходчика.

Нижняя оценка времени обхода для одного или ограниченного числа автоматов равна $\Omega(nm)$, где n — число вершин графа, а m — число дуг [[2]]. Если число автоматов не ограничено, нижняя оценка $\Omega(m)$.

Для того, чтобы автомат мог обходить любой конечный граф, требуется доступ по чтению/записи к неограниченной рабочей памяти, в которой накапливается информация о пройденной части графа. Если эта память — часть памяти автомата (машины), автомат не конечен на классе всех графов. Если автомат один, то существуют алгоритмы с оценкой $\Theta(nm)$ [[2]]. Если число автоматов больше одного, но ограничено, то распараллеливание ускоряет обход, но не меняет порядок времени обхода в наихудшем случае [[7]]. Если число автоматов не ограничено и все вычисления автоматов и передачи сообщений между ними выполняются не дольше (по порядку), чем проход дуги графа, то существуют алгоритмы с оценкой $\Theta(m)$. Это утверждение доказывается в данной статье в конце п.5.3.

Проблема возникает, когда граф не помещается в память машины, что эквивалентно конечности автомата. Есть два подхода.

Первый подход применим, когда рабочая память существует отдельно от памяти автоматов и реализуется на вершинах графа: автомат может писать/читать из текущей вершины символы конечного алфавита. Такой подход может применяться, например, для сети интернета, когда вершина — это узел сети, а проход по дуге — передача сообщения между узлами.

Для одного конечного автомата известен алгоритм с оценкой $\Theta(nm+n^2 loglogn)$ [[4]], а при повторном обходе $\Theta(nm+n^2 l(n))$, где $l(n)$ — число логарифмирований, при котором достигается соотношение $1 \le log(log...(n)...) < 2$ [[5]]. Отличие от нижней оценки $\Omega(nm)$ объясняется тем, что автомату бывает нужно «вернуться» в начало только что пройденной дуги.

Если конечных автоматов несколько, каждый из них может читать пометки в вершинах, оставленные другими автоматами, и обмениваться с ними сообщениями. Для двух автоматов оценка равна уже $\Theta(nm)$. Эти автоматы двигаются синхронно, кроме случая прохода по новой дуге. В этом случае один автомат (первый) идёт по дуге, а второй автомат остаётся на месте, ожидая сообщения от первого автомата. В этом сообщении указывается, нужно ли второму автомату оставаться на месте, поджидая первый автомат (для возвращения первого автомата в начало только что пройденной дуги), или, наоборот, двигаться вперёд, догоняя первый автомат (возвращения по дуге не требуется).

Второй подход применяется при тестировании, когда вершина графа — это состояние тестируемой системы, и автомат ничего не может в неё писать. Тогда рабочая память — это суммарная память коллектива конечных автоматов, обменивающихся сообщениями. Для k машин можно обходить графы в k раз большие, чем для одной машины. Если размер графа не ограничен, число автоматов в коллективе также должно быть не ограничено.

Этот подход впервые был применён в нашей работе [[9]], где предложен алгоритм обхода с оценкой $\boldsymbol{O}(m+n^2)$. В настоящей статье мы предлагаем алгоритм с улучшенной оценкой.

2. Формализация автоматов на графе

2.1. Номер выходящей дуги

Автомат на графе, начиная с начальной вершины графа, двигается по дугам, проходя некоторый маршрут. Обход выполнен, если по каждой дуге прошёл хотя бы один автомат. Когда автомат находится в вершине и хочет пройти по выходящей из неё дуге, он эту дугу должен указать. Для этого используется нумерация дуг, выходящих из вершины. Для детерминированных систем разные дуги, выходящие из одной вершины, имеют разные номера. Дуги, выходящие из вершины v нумеруются от 1 до полустепени выхода вершины (числа выходящих дуг): $1..d_{out}(v)$. Автомат указывает номер выходящей дуги. Однако для того, чтобы выходной алфавит автомата оставался конечным, полустепень выхода вершин графа должна быть ограничена сверху некоторой константой r.

Это ограничение легко снимается, если в каждой вершине v добавить $k=[d_{out}(v)/(r\text{-}1)]+1$ ячеек памяти, каждая из которых ассоциируется с r-1 выходящими дугами, кроме последней ячейки, которая может ассоциироваться с меньшим числом дуг. Эти ячейки связываются в цикл, который будем называть v-циклом. Для автомата добавляется *внутреннее* перемещение на следующую по v-циклу ячейку, которое автоматически настраивает граф на следующую порцию выходящих дуг. Тем самым, автомату, находящемуся в некоторой ячейке нужно идентифицировать внешнюю дугу только в пределах тех r-1 дуг, которые ассоциированы с этой ячейкой, или меньшего числа дуг для последней ячейки. При *внешнем* перемещении по дуге $(v`,v)$ автомат попадает в первую ячейку v-цикла (рис.1).

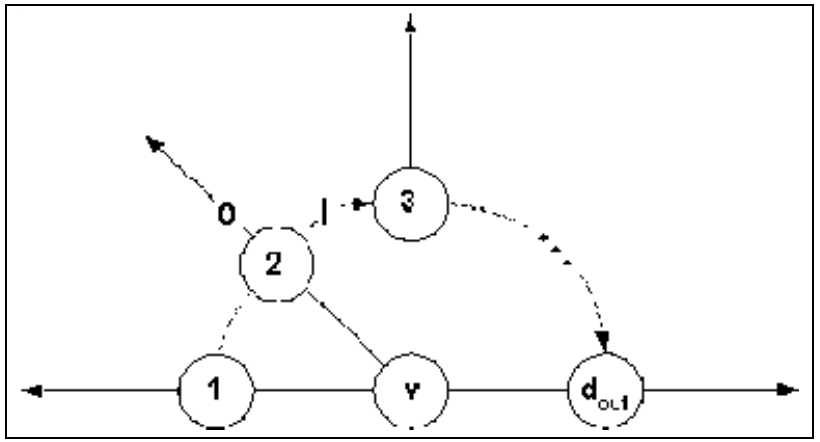

Рис 1. Вершина v и v-цикл ячеек (для r=2)

Это эквивалентно соответствующему преобразованию графа, когда каждая вершина v, в которой полустепень выхода больше r, заменяется на v-цикл вершин. В результате такого преобразования получается граф, в котором полустепень выхода каждой вершины v ограничена r. Дуги, заканчивавшиеся в вершине v, теперь будут заканчиваться в первой вершине v-цикла, а остальные вершины v-цикла отличаются тем, что в каждую из них входит ровно одна (внутренняя) дуга. В дальнейшем для простоты изложения мы будем рассматривать только такие графы, в которых из каждой вершины выходит не более r дуг.

Заметим, что замена вершины v на v-цикл вершин можно понимать как создание одним автоматом, оказавшимся в вершине v, цепочки автоматов — по одному на каждые r-1 выходящих дуг или меньшего числа дуг для последнего автомата в цепочке.

2.2. Идентификатор вершины

В процессе обхода графа в суммарной памяти автоматов создаются описания пройденных вершин и выходящих из них дуг. Описание вершины хранится в памяти одного автомата, который ниже будет называться регулятором вершины, и который регулирует движение автоматов, попавших в вершину, указывая им, по каким выходящим дугам им нужно двигаться. Для того, чтобы автомат понял, в какой вершине оказался, вершина снабжается уникальным идентификатором. Попав в вершину, автомат узнаёт её уникальный идентификатор и число выходящих из вершины дуг. По этому идентификатору автомат ищет регулятор вершины, опрашивая другие автоматы. Если регулятор не найден, опрашивающий автомат сам создаёт

описание вершины в своей памяти, становясь регулятором этой вершины. Поиск регулятора вершины не требуется в двух случаях: для терминальной вершины и для вершины только с одной входящей дугой.

Регулятор терминальной вершины не нужно искать, поскольку оказывающиеся в ней автоматы всё равно не могут двигаться дальше. Когда автомат проходит дугу и находит регулятор (или сам становится регулятором) конечной вершины дуги, он может его адрес сообщить регулятору начальной вершины дуги. Это делается для того, чтобы при повторном проходе дуги можно было не проводить поиск регулятора конечной вершины дуги. Поэтому, если известно, что у вершины есть только одна входящая дуга, регулятор этой вершины можно не искать: автомат, первым попавший в эту вершину сразу становится её регулятором. В этих двух случаях вершина может не иметь идентификатора, точнее она снабжается специальным *пустым* идентификатором. Заметим, что дополнительные вершины v-цикла, о которых шла речь в п.2.1, имеют по одной входящей дуге и поэтому могут снабжаться пустым идентификатором. Итак, непустой идентификатор обязана иметь каждая нетерминальная вершина, в которую входит более одной дуги. Остальные дуги могут иметь как пустой, так и непустой идентификатор. Все непустые идентификаторы вершин различны.

По техническим причинам мы будем считать, что идентификатор начальной вершины графа всегда не пустой.

Поскольку идентификатор вершины является входным символом автомата, для того, чтобы автомат был конечным, алфавит идентификаторов должен быть конечным. Но тогда для конечности автоматов мы должны рассматривать графы с ограниченным числом вершин с непустыми идентификаторами.

Для того, чтобы обойти это ограничение, предлагается рассматривать расширенный автомат (EFSM), в котором, кроме его состояния, имеется конечное число ячеек памяти, в которых могут храниться идентификаторы вершин и только они: в каждой ячейке по одному идентификатору. Эти ячейки будем называть ячейками идентификаторов (вершин).

Для идентификаторов вершин определены следующие операции и только они:

1. Скопировать идентификатор из указанной ячейки идентификаторов в сообщение, посылаемое другому автомату, как его параметр.
2. Скопировать идентификатор как параметр сообщения, полученного от другого автомата, в указанную ячейку идентификаторов.
3. Сравнить на равенство идентификаторы в двух указанных ячейках идентификаторов.

2.3. Адрес автомата и среда связи автоматов

Будем считать, что среда связи автоматов позволяет передавать сообщение от любого автомата любому указанному автомату за время, ограниченное сверху константой. При посылке сообщения автомат должен указать получателя сообщения как уникальный адрес автомата. Для конечного автомата число таких получателей должно быть конечным и ограниченным: адрес автомата должен пробегать конечный алфавит. Последнее ограничение приводит к тому, что число автоматов ограничено.

Чтобы обойти это ограничение, будем считать, что в расширенном автомате имеется конечное число ячеек памяти для хранения адресов автоматов. Эти ячейки будем называть ячейками адресов (автоматов). Тем самым, граф динамических связей автоматов — это конечный ориентированный граф с ограниченной полустепенью выхода вершин (автоматов).

Для адресов автоматов определены следующие операции и только они:

1. Послать сообщение автомату, адрес которого находится в указанной ячейке адресов.
2. Скопировать адрес из указанной ячейки адресов в посылаемое сообщение как его параметр.
3. Скопировать адрес как параметр полученного сообщения в указанную ячейку адресов.

Будем считать, что выделен один *пустой* адрес, который не может быть адресом никакого созданного автомата.

Мы будем предполагать выполнение следующих правил обмена сообщениями:

1. Для передачи сообщения есть один примитив «послать сообщение», обязательным параметром которого является адрес получателя.
2. Для получения сообщений есть один примитив «принять сообщение от любого отправителя».
3. Сообщения не теряются и не искажаются в среде передачи.
4. При передаче нескольких сообщений от одного и того же отправителя одному и тому же получателю нет обгона сообщений.

2.4. Сообщение

Сообщение, передаваемое от одного автомата другому автомату, является как входным, так и выходным символом автомата. Поэтому для того, чтобы автомат был конечным, сообщение должно быть символом из конечного алфавита. Но тогда сообщение не может содержать идентификаторы вершин и адреса автоматов, если размер графа или число автоматов не ограничены.

Чтобы обойти это ограничение, будем считать, что для расширенного автомата сообщение также «расширено»: оно состоит из *тега*, пробегающего конечный алфавит тегов, и ограниченного конечного числа параметров, которыми могут являться только идентификаторы вершин или адреса автоматов. В общем, это эквивалентно (неограниченной по длине) цепочке ограниченных по длине сообщений.

2.5. Порождение и уничтожение автоматов

Самый первый автомат порождается некоторым *внешним автоматом* и начинает свою работу с ожидания сообщения от этого внешнего автомата.

Для порождения остальных автоматов предлагается использовать сообщение ***создай автомат***, посылаемое внешнему автомату. В ответном сообщении ***автомат создан*** указывается адрес порождённого автомата.

Порождаемый автомат будет начинать работать с начальной вершины графа (если он вообще должен быть связан с графом). Это требование объясняется тем, что мы рассматриваем тестирование системы, которая не допускает произвольное клонирование, а только создание нового экземпляра системы в её начальном состоянии. Исключение составляет создание автомата, которому передаётся граф от другого автомата вместе с текущей вершиной. Передатчик теряет связь с графом.

Уничтожение автомата выполняется внешним автоматом. Для этого внешнему автомату посылается сообщение ***уничтожь автомат***, в котором указывает адрес автомата, который надо уничтожить. Ответ на это сообщение не предусматривается.

2.6. Взаимодействие с графом

Предлагается рассматривать граф, точнее его экземпляр для каждого взаимодействующего с ним автомата, как отдельный автомат графа. Такой автомат графа создаётся внешним автоматом, когда ему посылается сообщение ***создай граф***. В ответном сообщении ***граф создан*** содержится адрес (автомата) графа, идентификатор начальной вершины и число дуг, выходящих из начальной вершины. Созданный автомат графа находится в состоянии, которое соответствует начальной вершине графа.

Далее потребуется только одна операция по взаимодействию с графом, которая реализуется как сообщение, посылаемые автомату графа и ответное сообщение:

1. ***проход по дуге*** графа с параметром: номер выходящей дуги.
2. Ответное сообщение ***ответ на проход*** с параметрами: идентификатор вершины и число выходящих дуг. Имеются в виду идентификатор конечной вершины указанной дуги и число дуг, выходящих из этой вершины.

Для уничтожения графа внешнему автомату посылается сообщение ***уничтожь граф*** с параметром «адрес графа», который надо уничтожить. Ответ на это сообщение не предусматривается.

2.7. Замечание о ячейках автомата и параметрах сообщений

Далее, для простоты изложения, будем считать, что каждая ячейка памяти автомата или параметр сообщения может хранить как идентификатор вершины, так и адрес автомата. Мы будем просто говорить «ячейка» или «параметр».

Кроме того, для удобства изложения будут использоваться дополнительные ячейки и параметры фиксированного размера. Совокупность значений в этих ячейках является, фактически, расширением состояния автомата, а совокупность значений в этих параметрах является, фактически, расширением сообщения как дополнение к его тегу.

Автомат, память которого расширена такими ячейками, эквивалентен коллективу конечных автоматов, которые могут обмениваться между собой сообщениями.

3. Идея алгоритма

3.1. Граф

Рассматривается обход ориентированного графа с выделенной начальной вершиной и пронумерованными выходящими из каждой вершины дугами. Все вершины снабжены идентификаторами, как было описано в п.2.2. *Началом* и *концом* дуги для краткости будем называть, соответственно, начальную и конечную вершины дуги. *Терминальной вершиной* будем называть вершину графа, из которой не выходят дуги. *Терминальной дугой* будем называть дугу, заканчивающуюся в терминальной вершине.

После инициализации в каждый момент времени имеются:

- *Пройденный граф* — подграф графа, определяемый всеми пройденными дугами.
- *Пройденное дерево* — ориентированный от корня, которым является начальная вершина, остов подграфа, получаемого из пройденного графа удалением терминальных дуг. Такое пройденное дерево содержит все нетерминальные пройденные вершины пройденного графа.

В начале работы алгоритма, после инициализации, пройденный граф совпадает с пройденным деревом и состоит из одной начальной вершины графа.

В процессе работы алгоритма пройденный граф и пройденное дерево увеличиваются (добавляются новые дуги и вершины). Пройденное дерево, получающееся в конце работы, будем называть *законченным деревом*. Под

хордой будем понимать нетерминальную дугу, не принадлежащую законченному дереву; пройденная хорда является хордой пройденного дерева. Для данной вершины *входящей* дугой будем называть дугу законченного дерева, заканчивающуюся в этой вершине. У начальной вершины нет входящей дуги, а в любую другую вершину входит ровно одна дуга.

В конце работы алгоритма получается разметка графа: для каждой дуги в её начале указывается её тип: дуга законченного дерева, терминальная дуга или хорда.

3.2. Работа автоматов

Автомат работает в трёх режимах: генератор, регулятор, движок. Обход графа, то есть движение по его дугам, выполняют движки, регуляторы предназначены для управления перемещением движков по дугам, а генератор создаёт новые движки и связанные с ними копии графов.

Каждый создаваемый генератором движок предназначен для того, чтобы, начиная с начальной вершины графа, пройти путь в пройденном дереве, после чего пройти последовательность непройденных дуг, заканчивающуюся хордой или терминальной дугой. С каждой пройденной нетерминальной вершиной связан ровно один регулятор, который направляет движки, приходящие в эту вершину, по тем или иным выходящим из вершины дугам. Для этого движок, оказавшийся в данной вершине, спрашивает у регулятора этой вершины (посылает ему сообщение) ***куда идти***, а регулятор в ответном сообщении ***иди по дуге*** сообщает номер выходящей дуги. В некоторой ситуации (описанной ниже) движок оказывается «лишним» и останавливается, не пройдя ни одной непройденной дуги.

Автомат (копии) графа, связанный с движком «помнит» текущую вершину графа, в которой находится движок. Для прохода по дуге движок посылает автомату графа сообщение ***проход по дуге***, сообщая номер выходящей дуги, и получает в сообщении ***ответ на проход*** идентификатор конца дуги и число выходящих из него дуг.

Вначале извне создаётся генератор. После инициализации генератор одновременно становится регулятором начальной вершины, которая в этот момент времени является единственной пройденной вершиной и единственной вершиной пройденного дерева, а также создаёт первый движок, находящийся в начальной вершине графа.

Для управления обходом графа используются сообщения ***запрос*** и ***конец***. Такое сообщение посылается «по пройденной дуге» в обратном направлении: от автомата, находящегося в конце дуги, регулятору начала дуги. Сообщение ***запрос*** посылается регулятором вершины по входящей дуге и означает, что по этой дуге нужно послать один движок, когда будет такая возможность. Следующий движок по этой дуге можно посылать после того, как будет получен новый ***запрос***. Сообщение ***конец***, посылаемое по пройденной дуге, означает, что по этой дуге больше никогда не нужно посылать движки.

Сообщение ***конец*** посылает по хорде или терминальной дуге движок, который эту дугу проходит. Кроме того, сообщение ***конец*** посылает регулятор вершины по входящей дуге, если в пройденном дереве выше этой вершины нет непройденных дуг (точнее, нет вершин, из которых выходят непройденные дуги). Это происходит тогда, когда по всем дугам, выходящим из вершины, получено сообщение ***конец***. У начальной вершины нет входящей дуги, поэтому регулятор начальной вершины посылает сообщение ***конец*** внешнему автомату, что означает конец работы алгоритма. ***Запросы*** посылаются с некоторым опережением, поэтому может оказаться, что движку, пришедшему в вершину, некуда двигаться. Такой движок становится *ждущим* в вершине (запоминается регулятором вершины). Далее либо по выходящей дуге придёт ***запрос***, и движок будет послан по этой дуге, либо по всем выходящим дугам будут получены сообщения ***конец***, и тогда движок оказывается «лишним» – он должен остановиться.

Каждая дуга имеет в регуляторе её начала три состояния:

– *Активная* = дуга, по которой нужно посылать движки. Это может быть либо непройденная дуга (такие дуги всегда активны), либо дуга пройденного дерева, по которой пришёл ***запрос***, но движок по дуге ещё не послан.

– *Пассивная* = пройденная дуга, по которой пока не нужно посылать движки. Это дуга, по которой был послан движок, но после этого по дуге не приходили сообщения ***запрос*** или ***конец***.

– *Законченная* = пройденная дуга, по ней больше никогда не нужно посылать движки. Это дуга, по которой пришло сообщение ***конец***.

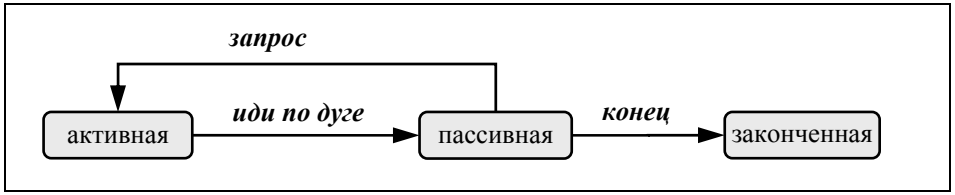

Рис 2. Схема изменения состояния дуги

На 0 изображена схема изменения состояния дуги под влиянием сообщений. В начале все дуги не пройдены, т.е. активные.

Регулятор посылает движки по активным выходящим дугам «по кругу». Для этого у регулятора есть номер текущей выходящей дуги *i*. Это дуга, по которой последний раз посылался движок, или *i*=0 в самом начале. Как сказано выше, движок, находясь в вершине пройденного дерева, спрашивает у

регулятора вершины, по какой дуге ему идти, посылая сообщение ***куда идти***. Регулятор корректирует номер i текущей дуги: новой текущей дугой становится следующая после дуги i активная дуга в цикле выходящих дуг, а если активных дуг больше нет, то i:=0. Если после коррекции i>0, регулятор посылает движку сообщение ***иди по дуге*** с номером дуги i, а сам по входящей дуге посылает ***запрос***. Поскольку у начальной вершины нет входящей дуги, регулятор начальной вершины не посылает сообщения ***запрос***, а генератор продолжает генерацию движков. Если i=0, регулятор запоминает адрес движка, движок становится ждущим, ***запрос*** не посылается. Если ждущий движок образуется в начальной вершине, генератор приостанавливает генерацию движков.

После того, как появляется ждущий движок, возможны два случая. 1) Регулятор в дальнейшем получает сообщение ***запрос*** с указанием дуги, по которой регулятор направляет ждущий движок (сообщение ***иди по дуге***), и сам посылает сообщение ***запрос*** регулятору начала входящей дуги. Поскольку у начальной вершины нет входящей дуги, регулятор начальной вершины не посылает сообщения ***запрос***, но генератор возобновляет генерацию движков. 2) Регулятор обнаруживает, что все выходящие дуги стали законченными (вершина терминальная или получено сообщение ***конец*** по последней незаконченной выходящей дуге). Регулятор посылает ждущему движку сообщение ***иди по дуге*** с нулевым номером дуги (это «лишний» движок и он останавливается) и сам посылает сообщение ***конец*** регулятору начала входящей дуги. Поскольку у начальной вершины нет входящей дуги, регулятор начальной вершины посылает сообщение ***конец*** внешнему автомату, и генератор прекращает генерацию движков.

Как создаются регуляторы? Рассмотрим подробнее работу автоматов, связанную с проходом движком непройденной ранее дуги a—i→b, ведущей из вершины a, имеющей в ней номер i и ведущей в вершину b.

- Если вершина b терминальная, для неё не создаётся регулятор. Движок посылает регулятору вершины a сообщение ***конец*** с признаком «терминальная вершина» и номером дуги i, а сам останавливается.
- Если вершина b нетерминальная, то движок должен узнать по идентификатору вершины, пройдена она или ещё нет, то есть имеется ли регулятор этой вершины. Вершина b заведомо не пройдена, если её идентификатор пуст (в такую вершину входит только одна дуга). В противном случае производится опрос регуляторов, описанный ниже. Итак:
  - Если вершина b уже пройдена, то движок прошёл по хорде. Движок посылает регулятору вершины a сообщение ***конец*** с признаком «хорда», номером дуги i и адресом регулятора вершины b, а сам останавливается.

  - Если вершина b ещё не пройдена, то движок сам становится регулятором вершины b и создаёт новый движок в текущей вершине b, передавая ему граф. После того, как этот новый движок будет создан и послан по выходящей дуге, новый регулятор вершины b посылает регулятору вершины a сообщение ***запрос*** с указанием номера дуги i и своим адресом как адресом регулятора вершины b.

Опрос регуляторов выполняется тогда, когда движок проходит по непройденной дуге и оказывается в нетерминальной вершине с непустым идентификатором. В этом случае ему нужно найти регулятор той вершины, в которой он оказался, или самому стать регулятором, если поиск не увенчался успехом. Этот поиск выполняется по идентификатору вершины: две вершины не могут иметь один и тот же непустой идентификатор. Для опроса регуляторов используется односторонний список регуляторов вершин с непустыми идентификаторами, регулятор начальной вершины (он же генератор) всегда находится в голове списка (напомним, что идентификатор начальной вершины всегда не пустой). Движок посылает по списку сообщение ***опрос***, в котором указывается (непустой) идентификатор вершины и адрес движка. Если регулятор вершины уже есть, он, получив такое сообщение, посылает ***ответ на опрос***, указывая движку свой адрес. Если регулятора вершины ещё нет, то сообщение ***опрос*** доходит до последнего в списке регулятора, который пересылает его движку, а сам ставит у себя ссылку по списку на этот движок. Получив обратно своё сообщение ***опрос***, движок сам становится регулятором – последним в списке, и создаёт новый движок, передавая ему свой адрес (как адрес регулятора текущей вершины) и адрес графа.

Замечание: Остановка движка означает, что движок уничтожает свою копию графа и самого себя.

4. Подробное описание алгоритма

Цикл работы автомата состоит в следующем:

- ожидание сообщения (любого сообщения от любого автомата),
- выполнение тех или иных действий в зависимости от полученного сообщения и состояния автомата.

Автомат имеет 4 управляющих состояния: 1) начальное состояние, 2) генератор (после инициализации, он же регулятор начальной вершины), 3) регулятор, 4) движок. Полное состояние расширенного автомата определяется его управляющим состоянием и содержанием ячеек памяти.

4.1. Структуры данных

4.1.1. Сообщения

Названия (теги) и параметры сообщений:

1. ***ты генератор***:
   - адрес внешнего автомата,
   - адрес генератора [то есть адрес получателя сообщения];
2. ***создай граф***:
   - адрес отправителя [сообщения];
3. ***граф создан***:
   - адрес [автомата копии] графа,
   - идентификатор начальной вершины,
   - число выходящих [из начальной вершины] дуг;
4. ***создай автомат***:
   - адрес отправителя [сообщения];
5. ***автомат создан***:
   - адрес созданного автомата;
6. ***ты движок***:
   - адрес движка [то есть адрес получателя сообщения],
   - адрес генератора,
   - адрес [автомата копии] графа,
   - идентификатор текущей вершины,
   - адрес регулятора текущей вершины;
7. ***куда идти***:
   - адрес отправителя [сообщения];
8. ***иди по дуге***:
   - номер выходящей дуги,
   - адрес регулятора конца выходящей дуги [может быть пустым];
9. ***проход по дуге***:
   - адрес отправителя [сообщения],
   - номер выходящей дуги;
10. ***ответ на проход***:
    - идентификатор вершины,
    - число выходящих [из этой вершины] дуг;
11. ***запрос***:
    - номер выходящей дуги [номер дуги, по которой в обратном направлении посылается сообщение, в начале дуги],
    - адрес регулятора конца выходящей дуги;
12. ***конец***:
    - номер выходящей дуги [номер дуги, по которой в обратном направлении посылается сообщение, в начале дуги],

    - адрес регулятора конца выходящей дуги [может быть пустым],
    - тип дуги [*дуга дерева*, *хорда*, *терминальная дуга*];
13. ***опрос***:
    - адрес движка,
    - идентификатор вершины;
14. ***ответ на опрос***:
    - адрес регулятора вершины;
15. ***уничтожь граф***:
    - адрес графа;
16. ***уничтожь автомат***:
    - адрес автомата.

Замечание: В сообщении ***запрос*** параметр «адрес регулятора конца выходящей дуги» лишний, если это повторное сообщение, посылаемое по этой дуге «в обратном направлении». Вместо этого можно было ввести сообщение с новым тегом ***новая вершина*** с такими же двумя параметрами, а в сообщении ***запрос*** оставить только один параметр: «номер выходящей дуги».

В сообщении ***конец***, которое посылает генератор внешнему автомату все параметры лишние. Вместо этого можно было бы ввести сообщение с новым тегом ***конец работы*** без параметров. Также в сообщении ***конец***, которое посылает движок, попавший в терминальную вершину, параметр «адрес регулятора конца выходящей дуги» всегда пустой. Вместо этого можно было бы ввести сообщение с новым тегом ***терминальная вершина***, в котором был только один параметр «номер выходящей дуги». Наконец, в сообщении ***конец***, которое посылается «в обратном направлении» по дуге пройденного дерева, параметр «адрес регулятора конца выходящей дуги» лишний, поскольку он уже известен регулятору начала дуги. Вместо этого можно было бы ввести сообщение с новым тегом ***хорда***, в котором были бы два параметра «номер выходящей дуги» и «адрес регулятора конца выходящей дуги», а сообщение ***конец*** посылать только «в обратном направлении» по дуге пройденного дерева с единственным параметром «номер выходящей дуги». Тем самым, вместо параметра «тип дуги» используются разные теги сообщений: ***терминальная вершина***, ***хорда***, ***конец***.

4.1.2. Ячейки автомата

- *собственный адрес автомата;*
- *адрес внешнего автомата;*
- *адрес* [автомата копии] *графа;*
- *адрес генератора;*
- *адрес ждущего движка;*
- *адрес следующего в списке регуляторов* [вершин с непустым идентификатором]*;*
- *идентификатор текущей вершины;*

- *адрес регулятора текущей вершины;*
- *номер входящей дуги* [пройденного дерева]*;*
- *адрес регулятора начала входящей дуги* [пройденного дерева]*;*
- *число выходящих дуг;*
- *номер текущей выходящей дуги;*
- список — *состояние выходящей дуги* (*i*), где *i*=1..*r;*
- список — *тип выходящей дуги* (*i*), где *i*=1..*r* [*дуга дерева*, *хорда*, *терминальная дуга*]*;*
- список — *адрес регулятора конца выходящей дуги* (*i*), где *i*=1..*r;*
- список — *рабочая ячейка* (*j*) , где *j*=1..3.

4.2. Работа автоматов

Работа начинается с того, что внешний автомат порождает один автомат, находящийся в начальном состоянии, которому посылает сообщение ***ты генератор***. Ниже мы описываем работу автомата при получении того или иного сообщения в зависимости от состояния автомата.

4.2.1. Сообщение ты генератор

Это сообщение может получить только автомат в начальном состоянии, который переходит в состояние «генератор».

Параметры сообщения переписываются в ячейки автомата:
адрес внешнего автомата, *адрес регулятора начала входящей дуги*
:= адрес внешнего автомата,
собственный адрес автомата, *адрес генератора*, *адрес регулятора текущей вершины*
:= адрес генератора.
Инициализируются ячейки:
адрес графа := *пустой*,
адрес ждущего движка := *пустой*.

Генератор создаёт экземпляр [автомата] графа, для чего посылает на *адрес внешнего автомата* сообщение ***создай граф*** с параметром:

адрес отправителя := *собственный адрес автомата*.

Обработка сообщения на этом заканчивается. В дальнейшем генератору может прийти только сообщение ***граф создан***.

4.2.2. Сообщение создай граф

Это сообщение может получить только внешний автомат и только от генератора.

Внешний автомат должен послать ответное сообщение ***граф создан***.

4.2.3. Сообщение граф создан

Это сообщение может получить только генератор в ответ на сообщение ***создай граф***.

Генератор проверяет, создаётся первый экземпляр графа или нет.

Если *адрес графа = пустой*, то создаётся первый экземпляр графа.

Параметры сообщения переписываются в ячейки автомата:

адрес графа := адрес графа,

идентификатор текущей вершины := идентификатор начальной вершины,

число выходящих дуг := число выходящих дуг.

Генератор проверяет, является ли начальная вершина терминальной.

Если *число выходящих дуг* = 0, то начальная вершина терминальная.

В этом случае генератор завершает работу алгоритма.

Для этого он сначала посылает на *адрес внешнего автомата* сообщение ***уничтожь граф*** с параметром: адрес графа := *адрес графа*.

Затем генератор посылает на *адрес внешнего автомата* сообщение ***конец***. Значения параметров в данном случае несущественны.

Обработка сообщения на этом заканчивается. Генератор останавливается.

Если *число выходящих дуг* > 0, то начальная вершина не терминальная.

Генератор становится регулятором начальной вершины, инициализируются ячейки регулятора:

адрес следующего в списке регуляторов := *пустой*,

номер текущей выходящей дуги := 0,

для каждого *i*=1..*число выходящих дуг* устанавливается

состояние выходящей дуги (*i*) := *активная*,

адрес регулятора конца выходящей дуги (*i*) := *пустой*.

Теперь генератор готов выполнять также функции регулятора начальной вершины.

Генератор посылает на *адрес внешнего автомата* сообщение ***создай автомат*** с параметром: адрес отправителя := *собственный адрес автомата*.

Обработка сообщения на этом заканчивается.

Если *адрес графа ≠ пустой*, то первый экземпляр графа был создан ранее.

Параметры сообщения переписываются в ячейки автомата:

адрес графа := адрес графа.

Остальные параметры игнорируются, поскольку они должны совпадать с теми, что были получены при создании первого экземпляра графа.

Генератор посылает на *адрес внешнего автомата* сообщение ***создай автомат*** с параметром: адрес отправителя := *собственный адрес автомата*.

Обработка сообщения на этом заканчивается.

4.2.4. Сообщение создай автомат

Это сообщение может получить только внешний автомат от генератора или движка, ставшего регулятором.

Внешний автомат должен послать ответное сообщение ***автомат создан***.

4.2.5. Сообщение автомат создан

Это сообщение является ответом на сообщение ***создай автомат***. Такой ответ может получить либо генератор, либо регулятор (движок, ставший регулятором). Созданный автомат предназначен для выполнения работы в режиме движка.

Параметры сообщения переписываются в ячейки автомата:

рабочая ячейка (1) := адрес созданного автомата.

Созданному автомату на адрес из *рабочая ячейка* (1) посылается сообщение ***ты движок*** с параметрами:

адрес движка := *рабочая ячейка* (1),
адрес генератора := *адрес генератора*,
адрес графа := *адрес графа*,
идентификатор текущей вершины := *идентификатор текущей вершины*,
адрес регулятора текущей вершины := *адрес регулятора текущей вершины*.
Обработка сообщения на этом заканчивается.

4.2.6. Сообщение ты движок

Это сообщение может получить автомат в начальном состоянии.

Автомат переходит в состояние движок.

Параметры сообщения переписываются в ячейки автомата:

собственный адрес автомата := адрес движка,
адрес генератора := адрес генератора,
адрес графа := адрес графа,
идентификатор текущей вершины := идентификатор текущей вершины,
адрес регулятора текущей вершины := адрес регулятора текущей вершины.

Движок посылает на *адрес регулятора текущей вершины* сообщение ***куда идти*** с параметром: адрес отправителя := *собственный адрес автомата*.

Обработка сообщения на этом заканчивается. В дальнейшем движок может получить только сообщение ***иди по дуге***.

4.2.7. Сообщение куда идти

Это сообщение может получить только регулятор (в том числе генератор как регулятор начальной вершины) и только от движка, оказавшегося в вершине этого регулятора.

Параметры сообщения переписываются в ячейки автомата:

рабочая ячейка (1) := адрес отправителя.

Регулятор корректирует *номер текущей выходящей дуги*. Для этого в цикле выходящих дуг (1..*число выходящих дуг*) после дуги с индексом *номер текущей выходящей дуги* ищется первый такой индекс *i*, что *состояние выходящей дуги* (*i*) = *активная*.

Если такая дуга найдена, то *номер текущей выходящей дуги* := *i*.
Иначе *номер текущей выходящей дуги* := *0*.

Далее регулятор проверяет, есть ли активная дуга.

Если *номер текущей выходящей дуги* > 0, то активная дуга есть.

Регулятор посылает в ответ по адресу из *рабочая ячейка* (1) сообщение ***иди по дуге*** с параметрами:

номер выходящей дуги := *номер текущей выходящей дуги*,

адрес регулятора конца выходящей дуги

:= *адрес регулятора конца выходящей дуги* (*номер текущей выходящей дуги*).

Регулятор корректирует

состояние выходящей дуги (*номер текущей выходящей дуги*) := *пассивная*.

Дальнейшее поведение зависит от управляющего состояния автомата.

Если это не генератор (не регулятор начальной вершины), то он посылает на *адрес регулятора входящей дуги* сообщение ***запрос*** с параметром:

номер выходящей дуги := *номер входящей дуги*.

Обработка сообщения на этом заканчивается.

Если это генератор, то он должен продолжить генерацию движков. Для этого генератор сначала должен создать новый экземпляр графа, поэтому он посылает на *адрес внешнего автомата* сообщение ***создай граф*** с параметром:

адрес отправителя := *собственный адрес автомата*.

Обработка сообщения на этом заканчивается.

Если *номер текущей выходящей дуги* = 0, то активных дуг нет.

Движок, спрашивавший ***куда идти***, становится ждущим. Для этого регулятор запоминает его: *адрес ждущего движка* := *рабочая ячейка* (1).

Заметим, что регулятор начальной вершины, то есть генератор приостанавливает генерацию движков, а регулятор другой вершины не посылает ***запрос***.

Обработка сообщения на этом заканчивается.

4.2.8. Сообщение иди по дуге

Это сообщение движок получает в ответ на сообщение ***куда идти***, которое он посылал регулятору той вершины, в которой сейчас находится.

Движок переписывает

адрес регулятора начала входящей дуги := *адрес регулятора текущей вершины*.

Параметры сообщения переписываются в ячейки автомата:

номер входящей дуги := номер выходящей дуги,
адрес регулятора текущей вершины := адрес регулятора конца выходящей дуги.

Движок проверяет, есть ли ему куда идти.

Если *номер входящей дуги = 0*, то движку идти некуда.

Движок должен уничтожить свою копию графа и сам себя.

Для этого он сначала посылает на *адрес внешнего автомата* сообщение ***уничтожь граф*** с параметром: адрес графа := *адрес графа*.

Затем движок посылает на *адрес внешнего автомата* сообщение ***уничтожь автомат*** с параметром: адрес автомата := *собственный адрес автомата*.

Обработка сообщения на этом заканчивается. Движок останавливается.

Если *номер входящей дуги ≠ 0*, то движок должен пройти по указанной дуге.

Для этого движок посылает на *адрес графа* сообщение ***проход по дуге*** с параметрами:
адрес отправителя := *собственный адрес автомата*,
номер выходящей дуги := *номер входящей дуги*.

Обработка сообщения на этом заканчивается. В дальнейшем движок может получить только сообщение ***ответ на проход***.

4.2.9. Сообщение проход по дуге

Это сообщение может получить только автомат графа и только от движка.

Автомат графа должен послать в ответ сообщение ***ответ на проход***.

4.2.10. Сообщение ответ на проход

Это сообщение движок получает от автомата графа в ответ на сообщение ***проход по дуге***.

Движок проверяет, прошёл он по новой (непройденной ранее) дуге или по старой дуге.

Если *адрес регулятора текущей вершины ≠ пустой*, то движок прошёл по старой дуге.

В этом случае движок игнорирует параметры сообщения: они ему не нужны, поскольку он знает адрес регулятора конца дуги как *адрес регулятора текущей вершины*.

Движок посылает на *адрес регулятора текущей вершины* сообщение ***куда идти*** с параметром: адрес отправителя := *собственный адрес автомата*.

Обработка сообщения на этом заканчивается. В дальнейшем движок может получить только сообщение ***иди по дуге***.

Если *адрес регулятора текущей вершины = пустой*, то движок прошёл по новой дуге.

Параметры сообщения переписываются в ячейки автомата:

идентификатор текущей вершины := идентификатор вершины,

число выходящих дуг := число выходящих дуг.

Движок проверяет, является ли текущая вершина терминальной.

Если *число выходящих дуг = 0*, то движок попал в терминальную вершину.

Движок посылает на *адрес регулятора начала входящей дуги* сообщение ***конец*** с параметрами:

номер выходящей дуги := *номер входящей дуги*,

адрес регулятора конца выходящей дуги := *пустой*,

тип дуги := *терминальная дуга*.

Движок должен уничтожить свою копию графа и сам себя.

Для этого он сначала посылает на *адрес внешнего автомата* сообщение ***уничтожь граф*** с параметром: адрес графа := *адрес графа*.

Затем движок посылает на *адрес внешнего автомата* сообщение ***уничтожь автомат*** с параметром: адрес автомата := *собственный адрес автомата*.

Обработка сообщения на этом заканчивается. Движок останавливается.

Если *число выходящих дуг > 0*, то движок попал в нетерминальную вершину.

Движок проверяет, не пуст ли *идентификатор текущей вершины*.

Если *идентификатор текущей вершины = пустой*, то в данную вершину входит только одна дуга – та, по которой прошёл движок. Поэтому, поскольку движок прошёл по новой дуге, эта вершина тоже новая.

Движок становится регулятором этой новой вершины, но не вставляется в список регуляторов, поскольку идентификатор вершины пустой.

Для этого меняется управляющее состояние автомата с «движок» на «регулятор» и инициализируются ячейки регулятора:

номер текущей выходящей дуги := 0,

для каждого *i*=1..*число выходящих дуг* устанавливается

состояние выходящей дуги (*i*) := *активная*,

адрес регулятора конца выходящей дуги (*i*) := *пустой*.

Движок, ставший регулятором, создаёт новый движок. Для этого он посылает на *адрес внешнего автомата* сообщение ***создай автомат*** с параметром: адрес отправителя := *собственный адрес автомата*.

Обработка сообщения на этом заканчивается. В дальнейшем движку, ставшему регулятором, может придти сообщение ***автомат создан***, а также сообщения ***опрос***, предназначенные ему как регулятору.

Если *идентификатор текущей вершины ≠ пустой*, движок выполняет опрос регуляторов для того, чтобы узнать, есть ли регулятор вершины, в которую он попал, или ещё нет.

Для этого движок посылает на *адрес генератора* (как первого регулятора в списке регуляторов) сообщение ***опрос*** с параметрами:

адрес движка := *собственный адрес автомата*,

идентификатор вершины := *идентификатор текущей вершины*.

Обработка сообщения на этом заканчивается. В дальнейшем движок может получить либо сообщение ***ответ на опрос***, либо сообщение ***опрос***.

4.2.11. Сообщение запрос

Это сообщение регулятор (в том числе генератор как регулятор начальной вершины) получает «в обратном направлении» по выходящей пассивной дуге от регулятора конца этой дуги.

Параметры сообщения переписываются в ячейки автомата:

рабочая ячейка (1) := номер выходящей дуги,
рабочая ячейка (*2*) := адрес регулятора конца выходящей дуги.

Регулятор проверяет, имеется ли у него ждущий движок.

Если *адрес ждущего движка = пустой*, то ждущего движка нет.

Дуга становится активной и меняется адрес регулятора конца дуги:

состояние выходящей дуги (*рабочая ячейка* (1)) := *активная*,
адрес регулятора конца выходящей дуги (*рабочая ячейка* (1)) := *рабочая ячейка* (*2*).

Обработка сообщения на этом заканчивается.

Если *адрес ждущего движка ≠ пустой*, то есть ждущий движок.

Ждущему движку на *адрес ждущего движка* посылается сообщение ***иди по дуге*** с параметрами:
номер выходящей дуги := *рабочая ячейка* (1),
адрес регулятора конца выходящей дуги := *рабочая ячейка* (*2*).
Дуга остаётся пассивной, а ждущего движка теперь нет:
адрес ждущего движка := *пустой*.

Дальнейшее поведение зависит от управляющего состояния автомата.

Если это не генератор (не регулятор начальной вершины), то он посылает на *адрес регулятора начала входящей дуги* сообщение ***запрос*** с параметрами:
номер выходящей дуги := *номер входящей дуги*,
адрес регулятора конца выходящей дуги := *собственный адрес автомата*.

Обработка сообщения на этом заканчивается.

Если это генератор, то он возобновляет генерацию движков. Для этого генератор сначала должен создать новый экземпляр графа, поэтому он посылает на *адрес внешнего автомата* сообщение ***создай граф*** с параметром:
адрес отправителя := *собственный адрес автомата*.

Обработка сообщения на этом заканчивается.

4.2.12. Сообщение конец

Это сообщение регулятор (в том числе генератор как регулятор начальной вершины) получает «в обратном направлении» по выходящей пассивной дуге от регулятора конца этой дуги или движка (в случае хорды или терминальной вершины).

Параметры сообщения переписываются в ячейки автомата:

рабочая ячейка (1) := номер выходящей дуги,
рабочая ячейка (*2*) := адрес регулятора конца выходящей дуги,
рабочая ячейка (*3*) := тип дуги.

Дуга становится законченной, корректируется адрес регулятора её конца и тип дуги:

состояние выходящей дуги (*рабочая ячейка* (1)) := *законченная*,
адрес регулятора конца выходящей дуги (*рабочая ячейка* (1)) := *рабочая ячейка* (*2*),
тип выходящей дуги (*рабочая ячейка* (1)) := *рабочая ячейка* (*3*).

Регулятор проверяет, есть ли ждущий движок.

Если *адрес ждущего движка = пустой*, то ждущего движка нет. В этом случае обработка сообщения заканчивается.
Если *адрес ждущего движка ≠ пустой*, то есть ждущий движок.
Регулятор проверяет, все ли выходящие дуги теперь законченные.
Замечание: Для оптимизации можно было бы использовать счетчик незаконченных выходящих дуг, который устанавливался бы равным числу выходящих дуг при инициализации ячеек в регуляторе и уменьшался бы на 1 при получении сообщения ***конец***.
Если для каждого *i*=1..*число выходящих дуг*
состояние выходящей дуги (*i*) = *законченная*, то все выходящие дуги законченные.
Ждущему движку на *адрес ждущего движка* посылается сообщение ***иди по дуге*** с параметром: номер выходящей дуги := 0. Остальные параметры несущественны.
По такому сообщению движок самоуничтожится.
Дальнейшее поведение зависит от управляющего состояния автомата.
Если это не генератор (не регулятор начальной вершины), то он посылает на *адрес регулятора начала входящей дуги* сообщение ***конец*** с параметрами:
номер выходящей дуги := *номер входящей дуги*,
адрес регулятора конца выходящей дуги := *собственный адрес автомата*,
тип дуги := *дуга дерева*.
Обработка сообщения на этом заканчивается.
Если это генератор, то он заканчивается работу алгоритма.
Для этого генератор посылает на *адрес внешнего автомата* сообщение ***конец***, параметры которого несущественны.
Обработка сообщения на этом заканчивается.
Если для некоторого *i*=1..*число выходящих дуг*
состояние выходящей дуги (*i*) ≠ *законченная*, то есть незаконченная выходящая дуга. Обработка сообщения на этом заканчивается.

4.2.13. Сообщение опрос

Это сообщение используется для поиска регулятора вершины по идентификатору вершины. Оно инициируется движком, прошедшим по новой (ранее непройденной) дуге и попавшим в нетерминальную вершину с непустым идентификатором. Движок направляет это сообщение генератору как регулятору начальной вершины и первому регулятору в списке регуляторов. Далее сообщение передаётся по списку регуляторов либо искомому регулятору вершины, если такой есть, либо, проходя весь список регуляторов, возвращается движку.

Обработка сообщения зависит от управляющего состояния автомата.

Если это регулятор (в том числе генератор как регулятор начальной вершины), то он проверяет идентификатор вершины.
Параметры сообщения переписываются в ячейки автомата:
рабочая ячейка (1) := адрес движка,
рабочая ячейка (*2*) := идентификатор вершины.
Регулятор проверяет, не он ли является искомым регулятором, то есть регулятором вершины с указанным идентификатором.
Если *идентификатор текущей вершины = рабочая ячейка* (*2*), то регулятор является искомым регулятором.
Регулятор посылает движку на адрес из *рабочая ячейка* (1) сообщение ***ответ на опрос*** с параметром: адрес регулятора вершины := *собственный адрес автомата*.
На этом обработка сообщения заканчивается.
Если *идентификатор текущей вершины ≠ рабочая ячейка* (1), то регулятор не является искомым регулятором.
Регулятор проверяет, не является ли он последним в списке регуляторов.
Если адрес следующего в списке регуляторов = 0, то регулятор последний в списке регуляторов. В этом случае регулятор делает движок, инициировавший запрос, последним в списке регуляторов:
адрес следующего в списке регуляторов := *рабочая ячейка* (1).
После этого регулятор пересылает на *адрес следующего в списке регуляторов* сообщение ***опрос*** с теми же параметрами:
адрес движка := *рабочая ячейка* (1),
идентификатор вершины := *рабочая ячейка* (*2*).
На этом обработка сообщения заканчивается.
Если это движок, то искомый регулятор не обнаружен и движок должен сам стать регулятором вершины.
Для этого меняется управляющее состояние автомата с «движок» на «регулятор» и инициализируются ячейки регулятора:
адрес следующего в списке регуляторов := *пустой*,
номер текущей выходящей дуги := 0,
для каждого *i*=1..*число выходящих дуг* устанавливается
состояние выходящей дуги (*i*) := *активная*,
адрес регулятора конца выходящей дуги (*i*) := *пустой*.
Движок, ставший регулятором, создаёт новый движок. Для этого он посылает на *адрес внешнего автомата* сообщение ***создай автомат*** с параметром:
адрес отправителя := *собственный адрес автомата*.
Обработка сообщения на этом заканчивается. В дальнейшем движку, ставшему регулятором, может придти сообщение ***автомат создан***, а также сообщения ***опрос***, предназначенные ему как регулятору, находящемуся в списке регуляторов.

4.2.14.Сообщение ответ на опрос

Это сообщение используется для поиска регулятора вершины по идентификатору вершины в случае, когда искомый регулятор найден. Найденный регулятор посылает сообщение ***ответ на опрос*** движку, инициировавшему опрос регуляторов. В этом случае движок прошёл по хорде и должен самоуничтожиться.

Адрес найденного регулятора как параметр сообщения переписывается в ячейку автомата:

рабочая ячейка (1) := адрес регулятора вершины.

Движок посылает на *адрес регулятора начала входящей дуги* сообщение ***конец*** с параметрами:

номер выходящей дуги := *номер входящей дуги*,
адрес регулятора конца выходящей дуги := *рабочая ячейка* (1),
тип дуги := *хорда*.

Движок должен уничтожить свою копию графа и сам себя.

Для этого он сначала посылает на *адрес внешнего автомата* сообщение ***уничтожь граф*** с параметром: адрес графа := *адрес графа*.

Затем движок посылает на *адрес внешнего автомата* сообщение ***уничтожь автомат*** с параметром: адрес автомата := *собственный адрес автомата*.

Обработка сообщения на этом заканчивается. Движок останавливается.

4.2.15. Сообщение уничтожь граф

Это сообщение может получить только внешний автомат.

Сообщение посылает генератор в случае, когда начальная вершина графа оказалась терминальной, или движок перед тем, как самоуничтожиться.

Внешний автомат освобождает память, занятую автоматом графа.

Ответное сообщение не предусмотрено.

4.2.16. Сообщение уничтожь автомат

Это сообщение может получить только внешний автомат.

Сообщение посылает движок для самоуничтожения.

Внешний автомат осовождает память, занятую движком.

Ответное сообщение не предусмотрено.

4.3. Описание графа в конце работы алгоритма

В конце работы алгоритма все движки и автоматы графов уничтожены. Сохраняются только регуляторы – по одному на каждую нетерминальную вершину графа.

Пройденный граф сопадает со всем графом (выполнен обход), а пройденное дерево содержит все нетерминальные вершины графа – мы назвали его законченным деревом.

Внешний автомат имеет адрес регулятора начальной вершины – это тот автомат, которому он в начале работы посылал сообщение ***ты генератор***. В регуляторе начальной вершины графа в ячейке *адрес регулятора начала входящей дуги* хранится адрес внешнего автомата.

В начале каждой дуги хранится ссылка на её конец, индексируемая номером этой дуги, а в конце каждой дуги законченного дерева хранится её номер в её начале и ссылка на её начало. Для этого используются регуляторы и их ячейки.

В регуляторе каждой нетерминальной вершины, кроме начальной вершины графа, имеется адрес регулятора начала входящей дуги законченного дерева в ячейке *адрес регулятора начала входящей дуги* и номер этой дуги в её начале, хранящийся в ячейке *номер входящей дуги*.

Также в регуляторе каждой нетерминальной вершины имеется число выходящих из этой вершины дуг в ячейке *число выходящих дуг* и описание всех выходящих из этой вершины дуг. Для каждого i=1.. *число выходящих дуг* в ячейке *тип выходящей дуги* (i) хранится тип i-ой выходящей дуги (дуга дерева, хорда или терминальная дуга), а в ячейке *адрес регулятора конца выходящей дуги* (i) хранится адрес регулятора конца i-ой выходящей дуги. Этот адрес пустой, если i-ая дуга терминальная.

5. Оценка сложности алгоритма

Для того, чтобы оценить сложность алгоритма, мы введём формальную модель обхода графа коллективом автоматов, докажем теорему о модели, установим связь модели с алгоритмом и оценим сложность алгоритма на основе теоремы о модели.

5.1. Модель обхода графа коллективом автоматов

5.1.1. Определение модели

Пусть задано дерево с начальной вершиной (корнем) и множеством вершин V и определены две функции $x{:}N\times V\to N$, $y{:}N\times V\to N$, где N – множество натуральных чисел. Будем обозначать: $x_{i,j}=x(i,j)$, $y_{i,j}=y(i,j)$. В вершинах дерева могут находиться точки, которые могут двигаться по дугам дерева. Пусть в момент времени t точка появилась в вершине j, причём в вершине j она является i-ой по счёту точкой (до неё в вершине j побывало i-1 точек). Эта точка задерживается в вершине j на время $y_{i,j}$. После этого точка может двигаться по дуге $j\to j`$, если в вершине $j`$ нет точки. Если таких дуг нет, точка i ожидает появления таких дуг. Если таких дуг несколько, выбирается одна из них недетерминированным образом. Если выбрана дуга $j\to j`$ в момент

времени $t`$, то точка $i$ появляется в вершине $j`$ в момент времени $t`$, и исчезает из вершины $j$ в момент времени $t`+x_{i,j}$. Если j корень, то $x_{i,j}=0$. В момент исчезновения точки из корня в корне появляется следующая точка. В начальный момент времени $t=0$ в корне появляется первая точка, в остальных вершинах точек нет.

Формализуем выбор дуг при движении точек с помощью функции $next:N\times V\times W\to V$, где $W\subseteq 2^V$. Если точка появляется в вершине j в момент времени t и является i-ой по счёту точкой в вершине j, а в ближайший после $t+y_{i,j}$ момент времени, когда множество «свободных» (в которых нет точек) концов выходящих из j дуг не пусто, это множество равно w, то $next(i,j,w)\in w$ и означает конец выбираемой выходящей из j дуги.

5.1.2. Обозначения и ограничения

- $m`$ – число дуг дерева. Число вершин дерева, очевидно, равно $m`+1$.
- Вершины дерева делятся на *простые* и *сложные*, что задаётся функцией $d:V\to\{0,1\}$: $d(j)=1$ тогда и только тогда, когда вершина j сложная. Будем обозначать $d_j=d(j)$.
- $n`$ – число сложных внутренних (отличных от листьев и корня) вершин дерева плюс 1. Тем самым, $n`>1$.
- Будем считать, что задержки ограничены сверху:
  - для $i>1$ $x_{i,j}\le\tau$,
  - для сложной j-ой вершины $y_{1,j}\le n`\tau$,
  - для $i>1$ или простой j-ой вершины $y_{i,j}\le\tau$,

  где τ – натуральное число, означающее длительность такта времени.
- Корень – простая вершина.
- $D`$ – максимальное число сложных нелистовых вершин на пути от корня до листа дерева.

5.2. Теорема о заполнении модели точками

Будем говорить, что модель заполнена точками, если в каждой вершине дерева находится точка. Обозначим через $t(m`,n`,D`)$ – максимальное время заполнения дерева точками для данных $m`$, $n`$ и $D`$.

Теорема 1: $t(m`,n`,D`)\le 4m`\tau+2(n`-1)D`\tau$.

Доказательство: Рассмотрим путь P в дереве от корня до листа и движение точек по этому пути. Перенумеруем вершины дерева (с соответствующим изменением функций x, y и $next$) так, чтобы вершины пути P от корня до листа получили номера $1,2,\ldots,k$. Будем говорить, что точка «уходит с пути» в вершине $j<k$, если при движении точек по дереву эта точка достигает вершины j на пути, из которой она уходит по дуге, не принадлежащей пути, т.е. уходит не в вершину $j+1$. Точка, попадающая в листовую вершину k, очевидно, в ней останавливается. Число точек, уходящих с пути, очевидно, равно $m`+1-k$. Будем говорить, что путь P заполняется точками в момент времени t_P, если в этот момент времени во всех вершинах пути P имеются точки, а в любой

предыдущий момент времени это не так. Оценим время заполнения пути P точками.

Перенумеруем все точки в порядке их генерации: 1,2,…. Очевидно, число сгенерированных точек в момент времени t_P не превосходит $m`+1$. Обозначим:

- $a(i,j)$ – максимальный номер точки, оказавшейся в вершине j, меньший i. Будем считать, что если все точки с номером меньше i не доходят до вершины j, то $a(i,j)=0$. Очевидно, что $a(i,j)\le i-1$.
- Также будем считать, что $t^+_{0,j}=0$.
- $f(j)$ – номер точки, которая первой попадает в вершину j. Очевидно, что $f(j)=i$ тогда и только тогда, когда $a(i,j)=0$ и $a(i+1,j)>0$, что эквивалентно $a(i,j)=0$ и $a(i+1,j)=i$.
- $t_{i,j}$ – время появления точки i в вершине j.
- $t^+_{i,j}$ – время исчезновения точки i из вершины j.
- d_j – равно 1, если вершина j сложная, и равно 0, если вершина j простая.
- $D_j=d_1+\ldots+d_j$ – число сложных вершин среди вершин $1..j$.
- $D_P=d_{k-1}$.
- $D_0=0$.
- $g(i,j)=r$, понимаемый как «локальный» номер точки i в вершине j, т.е. длина последовательности $i_1<i_2<\ldots<i_r$ такой, что $i_r=i$, для каждого $v=i_1,i_2,\ldots,i_r$ $a(v+1,j)=v$ и для каждого $w<i$ и отличного от $i_1,i_2,\ldots,i_r$ $a(w+1,j)<w$.
- $X_{i,j}=x_{g(i,j),j}$.
- $Y_{i,j}=y_{g(i,j),j}$.

Рассмотрим соотношения между временами, следующие из определения модели. Каждое такое соотношение выражает время прихода точки в вершину или время ухода точки из вершины через аналогичные времена, но, быть может, для других точек и вершин. Каждое такое соотношение, в зависимости от дополнительных условий, имеет вид $t=t`+\Delta$ или $t\le t`+\Delta$, где t и $t`$ имеют вид $t_{i,j}$ или $t^+_{i,j}$, а $\Delta$ – ограничение сверху на приращение времени. Мы будем говорить, что каждое такое соотношение определяет *шаг* перехода от времени $t$ к времени $t`$. С каждым временем $t_{i,j}$ или $t^+_{i,j}$ свяжем величину $2i+j+\pi$, где $\pi=0$ для $t_{i,j}$ и $\pi=1$ для $t^+_{i,j}$. Мы покажем приращение этой величины на каждом шаге, т.е. при переходе от t к $t`$, и ограничение сверху на приращение времени $\Delta\ge t-t`$.

0. Рассматриваем $t_{i,j}$ для $i=1$ и $j=1$.
   Время $t_{1,1}$ – это время прихода точки 1 в вершину 1, то есть момент появления первой точки в начале пути P, то есть в корне дерева. По определению модели это время равно 0.
   Соотношение 0: $t_{1,1}=0$.
   $2i+j+\pi=2\cdot1+1+0=3$.
1. Рассматриваем $t_{i,j}$ для $i>1$ и $j=1$.
   Время $t_{i,1}$ – это время прихода точки i в вершину 1, то есть момент появления точки i в начале пути P, то есть в корне дерева. По определению

модели любая точка, кроме первой, появляется в корне дерева в тот момент времени, когда предыдущая точка из этого корня уходит.

Соотношение 1: $t_{i,1}=t^{+}_{i-1,1}$.

Шаг $t_{i,1}\rightarrow t^{+}_{i-1,1}$.

Приращение $2i+j+\pi$: $[2(i-1)+1+1] - [2i+1+0] = -1$.

Приращение времени: = 0.

2. Рассматриваем $t_{i,j}$ для $j>1$ при условии, что точка i бывает в вершине j; это условие эквивалентно условию $a(i+1,j)=i$.

Если $j>1$, точка i появляется в вершине j в тот момент времени, когда выполнены два условия: 1) точка i прождала в предыдущей вершине j положенное время задержки $Y_{i,j-1}$, 2) из вершины j ушла предыдущая точка $a(i,j)$:

$t_{i,j}=max\{t_{i,j-1}+Y_{i,j-1},t^{+}_{a(i,j),j}\}$.

Заметим, что если предыдущей точки нет, то есть точка i появляется первой в вершине j, то требуется выполнение только одного условия 1. Но в этом случае $a(i,j)=0$, поэтому по определению $t^{+}_{a(i,j),j}=0$ и, следовательно, $t_{i,j}=max\{t_{i,j-1}+Y_{i,j-1},0\}=t_{i,j-1}+Y_{i,j-1}$.

2.1. Рассматриваем случай, когда точка i не первая в вершине j-1, что эквивалентно условию $g(i,j-1)>1$.

Тогда по ограничению $Y_{i,j-1}\le\tau$.

2.1.1. Рассматриваем случай $t_{i,j-1}+Y_{i,j-1} \ge t^{+}_{a(i,j),j}$.

Соотношение 2.1.1: $t_{i,j} \le t_{i,j-1}+\tau$.

Шаг $t_{i,j}\rightarrow t_{i,j-1}$.

Приращение $2i+j+\pi$: $[2i+(j-1)+0] - [2i+j+0] = -1$

Приращение времени: $\le \tau$.

2.1.2. Рассматриваем случай $t_{i,j-1}+Y_{i,j-1} < t^{+}_{a(i,j),j}$.

Соотношение 2.1.2: $t_{i,j} = t^{+}_{a(i,j),j}$.

Шаг $t_{i,j}\rightarrow t^{+}_{a(i,j),j}$.

Приращение $2i+j+\pi$: $[2a(i,j)+j+1] - [2i+j+0] \ge [2(i-1)+j+1] - [2i+j+0] = -1$.

Приращение времени: = 0.

2.2. Рассматриваем случай, когда точка i первая в вершине j-1, что эквивалентно условию $g(i,j-1)=1$.

В этом случае по ограничению $Y_{i,j-1}\le\tau+(n`-1)d_{j-1}\tau$.

Также в этом случае до точки i никакая точка не была в вершине j: $a(i,j)=0$.

Поэтому $max\{t_{i,j-1}+Y_{i,j-1},t^{+}_{a(i,j),j}\} = max\{t_{i,j-1}+Y_{i,j-1}, 0\} = t_{i,j-1}+Y_{i,j-1}$.

Соотношение 2.2: $t_{i,j} \le t_{i,j-1}+\tau+(n`-1)d_{j-1}\tau$.

Шаг $t_{i,j}\rightarrow t_{i,j-1}$.

Приращение $2i+j+\pi$: $[2i+(j-1)+0] - [2i+j+0] = -1$

Приращение времени: $\le \tau+(n`-1)d_{j-1}\tau$.

3. Рассматриваем $t^{+}_{i,j}$ для $j>1$ при условии, что точка i бывает в вершине j; это условие эквивалентно условию $a(i+1,j)=i$.

3.1. Рассматриваем случай, когда точка i бывает в вершине j+1, что эквивалентно $a(i+1,j+1)=i$.

Точка i покидает вершину j через интервал времени $X_{i,j+1}$ после её появления в следующей вершине j+1.

$t^{+}_{i,j}=t_{i,j+1}+X_{i,j+1}=max\{t_{i,j}+Y_{i,j},t^{+}_{a(i,j+1),j+1}\}+X_{i,j+1}$.

3.1.1. Рассматриваем случай, когда точка i не первая в вершине j, что эквивалентно условию $g(i,j)>1$.

Тогда по ограничению $Y_{i,j}\le\tau$. Также по ограничению $X_{i,j+1}\le\tau$.

3.1.1.1. Рассматриваем случай $t_{i,j}+Y_{i,j} \ge t^{+}_{a(i,j+1),j+1}$.

Соотношение 3.1.1.1: $t^{+}_{i,j} \le t_{i,j}+\tau+\tau$.

Шаг $t^{+}_{i,j}\rightarrow t_{i,j}$.

Приращение $2i+j+\pi$: $[2i+j+0] - [2i+j+1] = -1$

Приращение времени: $\le 2\tau$.

3.1.1.2. Рассматриваем случай $t_{i,j}+Y_{i,j} < t^{+}_{a(i,j+1),j+1}$.

Соотношение 3.1.1.2: $t^{+}_{i,j} = t^{+}_{a(i,j+1),j+1}+\tau$.

Шаг $t^{+}_{i,j}\rightarrow t^{+}_{a(i,j+1),j+1}$.

Приращение $2i+j+\pi$: $[2a(i,j+1)+j+1] - [2i+j+1] \ge [2(i-1)+j+1] - [2i+j+1] = -2$.

Приращение времени: = τ.

3.1.2. Рассматриваем случай, когда точка i первая в вершине j, что эквивалентно условию $g(i,j)=1$.

В этом случае по ограничению $Y_{i,j}\le\tau+(n`-1)d_j\tau$. Также по ограничению $X_{i,j+1}\le\tau$.

Также в этом случае до точки i никакая точка не была в вершине j+1: $a(i,j+1)=0$.

Поэтому $max\{t_{i,j}+Y_{i,j},t^{+}_{a(i,j+1),j+1}\}+X_{i,j+1} = max\{t_{i,j}+Y_{i,j},0\}+X_{i,j+1} = t_{i,j}+Y_{i,j}+X_{i,j+1}$.

Соотношение 3.1.2: $t^{+}_{i,j} \le t_{i,j}+\tau+(n`-1)d_j\tau+\tau$.

Шаг $t^{+}_{i,j}\rightarrow t_{i,j}$.

Приращение $2i+j+\pi$: $[2i+j+0] - [2i+j+1] = -1$

Приращение времени: $\le 2\tau+(n`-1)d_j\tau$.

3.2. Рассматриваем случай, когда точка i доходит до вершины j, что эквивалентно $a(i+1,j)=i$, в которой уходит с пути, что эквивалентно $a(i+1,j+1)<i$.

3.2.1. Рассмотрим случай, когда $t_{i,j}+Y_{i,j}\ge t^{+}_{a(i,j+1),j+1}$.

Это случай, когда точка i доходит до вершины j в момент времени $t_{i,j}$ и после задержки $Y_{i,j}$, то есть в момент времени $t_{i,j}+Y_{i,j}$, следующая вершина j+1 на пути свободна. В этот момент времени точка i может двигаться из вершины j, поэтому она должна покинуть вершину j. В рассматриваемом случае точка уходит с пути.

Имеем $t^{+}_{i,j}=t_{i,j}+Y_{i,j}$.

3.2.1.1. Рассмотрим случай, когда точка i не первая в вершине j, что эквивалентно $g(i,j)>1$.

Тогда по ограничению $Y_{i,j} \le \tau$.

Соотношение 3.2.1.1: $t^+_{i,j} \le t_{i,j}+\tau$.

Шаг $t^+_{i,j} \to t_{i,j}$.

Приращение $2i+j+\pi$: $[2i+j+0] - [2i+j+1] = -1$

Приращение времени: $\le \tau$.

3.2.1.2. Рассмотрим случай, когда точка i первая в вершине j, что эквивалентно $g(i,j)=1$.

Тогда по ограничению $Y_{i,j} \le \tau+(n`-1)d_j\tau$.

Соотношение 3.2.1.2: $t^+_{i,j} \le t_{i,j}+\tau+(n`-1)d_j\tau$.

Шаг $t^+_{i,j} \to t_{i,j}$.

Приращение $2i+j+\pi$: $[2i+j+0] - [2i+j+1] = -1$

Приращение времени: $\le \tau+(n`-1)d_j\tau$.

3.2.2. Рассмотрим случай, когда $t_{i,j}+Y_{i,j} < t^+_{a(i,j+1),j+1}$.

Это случай, когда точка i доходит до вершины j в момент времени $t_{i,j}$ и после задержки $Y_{i,j}$, то есть в момент времени $t_{i,j}+Y_{i,j}$, следующая вершина $j+1$ на пути занята. Тогда точка i обязана покинуть вершину j после того, как следующая вершина $j+1$ освободится, но уйти с пути она может и раньше.

Соотношение 3.2.2: $t^+_{i,j} \le t^+_{a(i,j+1),j+1}$.

Шаг $t^+_{i,j} \to t^+_{a(i,j+1),j+1}$.

Приращение $2i+j+\pi$: $[2a(i,j+1)+j+1+1] - [2i+j+1] \le [2(i-1)+j+1+1] - [2i+j+1] = -1$.

Приращение времени: ≤ 0.

Прежде всего, заметим, что для каждых i, j при условии, что точка i генерируется и достигает вершины j, для времени $t_{i,j}$ можно сделать шаг $t_{i,j} \to t`$, определяемый каким-то соотношением, а в случае $j<k$ для времени $t^+_{i,j}$ также можно сделать шаг $t^+_{i,j} \to t`$, определяемый каким-то соотношением. На каждом шаге величина $2i+j+\pi$ уменьшается не менее, чем на 1, а минимальное значение 3 эта величина принимает для $t_{1,1}=0$. Поэтому для $t_{m`+1,1}$ число шагов не превосходит $[2(m`+1)+1+0]-3 = 2m`$.

Приращение времени на шаге ограниченно сверху: А) числом 2τ или В) числом $2\tau+(n`-1)d_j\tau$. Для $d_j=0$ имеем $2\tau+(n`-1)d_j\tau = 2\tau$, а для $d_j=1$ имеем $2\tau+(n`-1)d_j\tau = (n`+1)\tau$. Поскольку $n` \ge 1$, $(n`+1)\tau \ge 2\tau$, то есть в варианте В ограничение сверху не меньше, чем в варианте А.

Теперь посмотрим, в каких соотношениях используется d_j в приращении времени. Это соотношения 2.2, 3.1.2 и 3.2.1.2. Соотношение 2.2 определяет шаг $t_{i,j+1} \to t_{i,j}$ при условии, что точка i бывает в вершине $j+1$ и является первой точкой в вершине j. Соотношения 3.1.2 и 3.2.1.2 определяют шаг $t^+_{i,j} \to t_{i,j}$ при условии, что точка i первой приходит в вершину j. Поскольку в каждой вершине только одна точка является первой, d_j используется не более двух

раз: на шаге $t_{i,j+1} \to t_{i,j}$ и на шаге $t^+_{i,j} \to t_{i,j}$ при условии в обоих случаях, что точка i первая в вершине j. Следовательно, вариант В для приращения времени возможен не более, чем на двух шагах для данного j. Поэтому общее число шагов с вариантом В не превосходит $2D_P$. Поскольку ограничение сверху для варианта В не меньше, чем для варианта А, общее приращение времени на всех шагах для $t_{m`+1,1}$ не превосходит $(2m`-2D_P)2\tau + 2D_P(n`+1)\tau = 4m`\tau + 2(n`-1)D_P\tau$.

Путь P заполняется точками в момент времени t_P, причём в этот момент времени число сгенерированных точек не превосходит $m`+1$. Это заполнение происходит тогда, когда последняя сгенерированная точка появляется в вершине 1, что происходит не позже, чем появление точки $m`+1$ в вершине 1. Иными словами, $t_P \le t_{m`+1,1}$.

Заполнение пути P не обязательно означает заполнение всего дерева, поскольку после заполнения пути точки могут уходить с пути, если дерево не заполнено. Однако существует такой путь в дереве, момент заполнение которого совпадает с моментом заполнения всего дерева.

В каждой вершине дерева отметим выходящую дугу, по которой последний раз двигалась точка до заполнения дерева. Путь от корня до листа по отмеченным дугам будем называть *последним путём* и обозначать L.

Лемма 1: Последний путь заполняется одновременно с заполнением всего дерева.

Доказательство:

Рассмотрим конечный отрезок последнего пути, полностью заполненный точками.

Покажем, что эти точки не уходят с пути.

Действительно, допустим противное: из какой-то вершины отрезка точка уходит с пути. Рассмотрим последнюю такую вершину j. Эта вершина j не может быть листовой, так как из листовых вершин точки не уходят. Уход точки с пути из вершины пути означает, что точка передвинулась из этой вершины по дуге дерева, не лежащей на этом пути. Но тогда получится, что последняя дуга, по которой точка вышла из вершины j не лежит на последнем пути. Мы пришли к противоречию, следовательно, утверждение верно.

Следовательно, в тот момент времени, когда последний путь оказывается заполненным, ни одна точка из его вершины не уходит с пути. А тогда всё дерево заполнено, так как в противном случае была бы незаполненная вершина, не лежащая на последнем пути, и для её заполнения самая верхняя точка, общая для пути в эту вершину и последнего пути, должна была бы уйти с последнего пути. Поэтому в тот момент времени, когда последний путь оказывается заполненным, всё дерево также оказывается заполненным.

Лемма 1 доказана.

Таким образом, имеем: $t(m`,n`,D`) = t_L \le t_{m`,1} \le 4m`\tau+2(n`-1)D_L\tau \le 4m`\tau+2(n`-1)D`\tau$.
Теорема 1 доказана.

5.3. Прагматика: связь модели с алгоритмом. Сложность алгоритма

- Дерево модели получается из исходного графа следующим образом. Берём законченное дерево (пройденное дерево в конце работы алгоритма) и для каждой хорды добавляем новую вершину, которую делаем новым концом хорды. После этого добавляем терминальные дуги графа.
- m – число дуг в исходном графе. Очевидно, $m`=m$.
- Простая вершина – это терминальная вершина, вершина с пустым идентификатором или начальная вершина, т.е. корень дерева модели. Корень считается простой вершиной, поскольку переход движка по дуге, ведущей в начальную вершину, в модели превращается в переход точки в листовую вершину, а движок, сгенерированный в начальной вершине, не производит опрос регуляторов.
  Сложная вершина – это любая другая вершина, т.е. нетерминальная вершина с непустым идентификатором, отличная от корня. В такой вершине происходит опрос регуляторов.
- n – число нетерминальных вершин с непустым идентификатором. Очевидно $n`=n$, если начальная вершина нетерминальная и имеет непустой идентификатор, и $n`=n+1$, если начальная вершина терминальная или имеет пустой идентификатор. Поэтому всегда $n`\le n+1$.
- D – максимальное число нетерминальных вершин с непустым идентификатором на пути в графе от начальной вершины, не считая самой начальной вершины. Очевидно, что $D`=D$.
- Точка в модели соответствует движку. Отличие только в том, что движок, прошедший по терминальной дуге (нулевое число выходящих дуг как параметр сообщения ***ответ на проход***) или хорде (сообщение ***ответ на опрос***), а также ждущий движок, получивший сообщение ***иди по дуге*** с нулевым номером дуги, останавливается и уничтожается, а в модели точка остаётся на месте.
- Появление точки в корне дерева модели соответствует началу генерации движка, а появление точки в другой вершине – началу перемещения движка по дуге, ведущей в эту вершину (сообщение ***иди по дуге***).
- Исчезновение точки из корня дерева модели соответствует началу перемещения движка из начальной вершины по выходящей дуге, что совпадает с началом генерации следующего движка.
  Исчезновение точки из некорневой вершины дерева модели соответствует получению регулятором начала входящей в эту вершину дуги сообщения ***запрос*** по этой дуге, что происходит после того, как по этой дуге пошёл движок (сообщение ***иди по дуге***).

- $x_{1,1}=0$ (корень дерева модели имеет номер $j=1$) означает, что генерация первого движка начинается в начальный момент времени 0.
- $x_{i,1}=0$ (корень дерева модели имеет номер $j=1$) для $i>1$ означает, что генерация движка начинается в тот момент времени, когда предыдущий движок пошёл по дуге из начальной вершины: регулятор начальной вершины не посылает генератору сообщение ***запрос***; вместо этого генератор, совпадающий с регулятором начальной вершины, сразу начинает генерировать следующий движок.
- $x_{i,j}$, если $j>1$, т.е. вершина j не корень дерева модели, означает время передачи из регулятора вершины j в регулятор предшествующей по дереву вершины сообщения ***запрос***.
- $y_{1,1}$ означает время генерации первого движка, т.е. время передачи следующей цепочки сообщений:
  - ***ты генератор, создай граф***, ***граф создан***,
    создай автомат, ***автомат создан***, ***куда идти***.
- $y_{i,1}$, где $i>1$, означает время генерации i-го движка, т.е. время передачи следующей цепочки сообщений:
  - ***создай граф***, ***граф создан***,
    создай автомат, ***автомат создан***, ***куда идти***.
- $y_{1,j}$ для $j>1$ означает время движения 1-го движка по дуге плюс опрос регуляторов, если конец этой дуги – сложная вершина, т.е. время передачи одной из следующих цепочек сообщений:
  - терминальная вершина j:
    иди по дуге, ***проход по дуге***, ***ответ на проход***;
  - идентификатор нетерминальной вершины j пустой:
    иди по дуге, ***проход по дуге***, ***ответ на проход***,
    создай автомат, ***автомат создан***, ***куда идти***;
  - идентификатор нетерминальной вершины j не пустой:
    иди по дуге, ***проход по дуге***, ***ответ на проход***, ***опрос***…***опрос***,
    создай автомат, ***автомат создан***, ***куда идти***.
- $y_{i,j}$ для $j>1$ и $i>1$ означает время прохода i-го (не первого) движка в вершину j, т.е. время передачи следующей цепочки сообщений (без опроса):
  иди по дуге, ***проход по дуге***, ***ответ на проход***, ***куда идти***.
- Задержка τ означает максимальное время передачи любой указанной выше цепочки сообщений, кроме той, где участвуют опросы.
- Задержка $n`\tau$ – это ограничение сверху на передачу цепочки сообщений с опросом списка регуляторов. В список попадает регулятор нетерминальной вершины с непустым идентификатором, а также регулятор начальной вершины. Число таких регуляторов равно числу сложных нелистовых вершин дерева модели плюс 1, т.е $n`$. В модели не учитываются задержки при опросе регуляторов в листовых вершинах дерева, т.е. задержки при опросе после прохода движка по хорде, когда он вместо сообщения ***опрос*** получает сообщение ***ответ на опрос***. Если же

движок проходит в новую нетерминальную вершину с непустым идентификатором, то он получит сообщение ***опрос***, после чего становится регулятором этой вершины и вставляется в список регуляторов. А это означает, что в этом случае при опросе число регуляторов в списке меньше, по крайней мере на 1, т.е. не превосходит $n`$-1. Цепочка с опросом регуляторов всегда содержит подпоследовательность сообщений ***иди по дуге***, ***проход по дуге***, ***ответ на проход***, ***опрос*** (***опрос***, который посылает движок), ***создай автомат***, ***автомат создан***, ***куда идти***, время передачи которых тоже ограничим временем τ. Остальная часть цепочки – это последовательность сообщений ***опрос***, посылаемых регуляторами в списке, длина которого не превосходит $n`$-1. Поэтому задержка на передачу сообщений всей цепочки не превосходит $\tau+(n`-1)\tau=n`\tau$.

На основе рассмотренной выше связи модели с алгоритмом можно заключить, что в момент заполнения модели точками в каждой вершине исходного графа либо 1) находится движок, который выполняет опрос регуляторов, либо 2) находится ждущий движок, либо 3) движок находился раньше, а потом был уничтожен. В случае 1 движок прошёл по хорде и находится в её конце, который соответствует листовой вершине дерева модели. В случае 3 регулятор вершины уже послал сообщение ***конец*** по входящей дуге, если это не начальная вершина, или алгоритм закончил свою работу посылкой сообщения ***конец*** внешнему автомату. После момента заполнения модели точками на исходном граф выполняются следующие действия: A) движки, выполняющие опрос регуляторов, завершают эти опросы и посылают сообщения ***конец***; B) сообщения ***конец*** распространяются по цепочке регуляторов на пути дерева к регулятору начальной вершины. Действия A требуют время, не превосходящее время опроса регуляторов после прохода по хорде, когда в списке регуляторов могут оказаться регуляторы всех нетерминальных вершин с непустым идентификатором и регулятор начальной вершины, т.е. не превосходящее $(n+1)\tau$. Распространение сообщений ***конец*** происходит по пути в законченном дереве от листьев до корня. Длина такого пути не превосходит числа дуг графа m, а время передачи одного сообщения можно считать не превосходящим τ. Поэтому действия B требуют время, не превосходящее $m\tau$.

Таким образом, учитывая теорему 1, алгоритм заканчивает работу за время, не более $[4m`\tau+2(n`-1)D`\tau]+(n+1)\tau+m\tau \le 4m\tau+2nD\tau+(n+1)\tau+m\tau$. Поскольку, очевидно, $n\le m+1$. Поэтому $4m\tau+2nD\tau+(n+1)\tau+m\tau \le 4m\tau+2nD\tau+(m+1+1)\tau+m\tau = 6m\tau+2nD\tau+2\tau = \boldsymbol{O}(m+nD)$, где

- m – число дуг в исходном графе,
- n – число нетерминальных вершин с непустым идентификатором,
- D – максимальное число нетерминальных вершин с непустым идентификатором на пути в графе от начальной вершины, не считая самой начальной вершины.

Во введении упоминалось утверждение о том, что если число автоматов не ограничено и все вычисления автоматов и передачи сообщений между ними выполняются не дольше (по порядку), чем проход дуги графа, то существуют алгоритмы, работающие $\boldsymbol{\Theta}(m)$ времени. В этом случае мы можем считать, что все вершины простые, т.е. нет опроса регуляторов, длительность которого зависит от n. Поэтому в этом случае из доказанной сложности алгоритма следует оценка $6m\tau+2\tau = \boldsymbol{O}(m)$. Нижняя оценка времени работы любого алгоритма обхода равна $\boldsymbol{\Omega}(m)$ при условии, что за один такт можно сгенерировать не более одного автомата и каждый автомат за один такт проходит не более одной дуги. Поэтому получается оценка $\boldsymbol{\Theta}(m)$.

Для того, чтобы оценить время работы алгоритма снизу, нужно предположить, что задержки $x_{i,j}$ и $y_{i,j}$ ограничены снизу. Для любого i и j>1 можно считать $x_{i,j}\ge 1$. Для i>1 и любого j можно считать, что $y_{i,j}\ge 1$. Если вершина j простая, то можно считать, что $y_{1,j}\ge 1$. А если вершина j сложная, то $y_{1,j}$ ограничено снизу 1 плюс число сложных вершин, которых достигли точки к тому моменту времени, когда в вершину j приходит первая точка. Пусть эти условия выполнены и, кроме того, $m+D\ge 2n+1$. Рассмотрим граф, изображённый на 0. Мы оставляем без доказательства утверждение о том, что на этом графе алгоритм работает $\boldsymbol{\Omega}(m+nD)$.

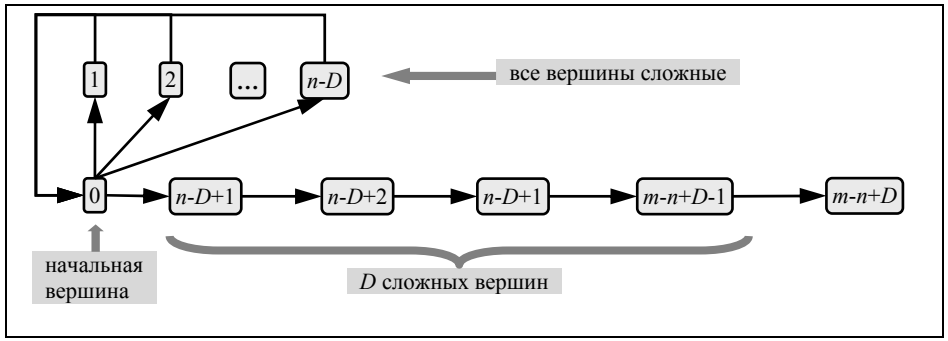

Рис. 3. Пример графа, где алгоритм работает время $\boldsymbol{\Omega}(m+nD)$

6. Возможные оптимизации

6.1. Хорда

В рассмотренном выше алгоритме движок, прошедший по хорде (и узнавший об этом в результате опроса регуляторов), прекращает свою работу и уничтожается. Вместо этого можно было бы оптимизировать алгоритм, разрешив такому движку продолжать работу по обходу графа.

Движок узнаёт о том, что он прошёл по хорде, когда получает от регулятора конца хорды сообщение ***ответ на опрос***, в котором сообщается адрес этого регулятора. После этого движок мог бы послать этому регулятору сообщение ***куда идти***, и, получив ответное сообщение ***иди по дуге***, идти по указанной выходящей дуге, если в этом ответном сообщении указан ненулевой номер выходящей дуги. Нулевой номер выходящей дуги в сообщении ***иди по дуге*** регулятор указывает в том случае, когда идти некуда: все выходящие из вершины дуги пассивные и уже есть ждущий в этой вершине движок.

При такой модификации возникает следующая проблема. Когда движок попадает в вершину и не становится ждущим движком, регулятор вершины посылает по этой дуге в обратном направлении сообщение ***запрос***. До модификации движок, прошедший хорду, не посылает регулятору вершины сообщение ***куда идти***, поэтому регулятор не посылает сообщение ***запрос***. После модификации регулятор будет стараться направить по выходящим дугам все движки, приходящие в вершину, включая те, что приходят по хорде. Это может привести к появлению «лишних» сообщений ***запрос***, которые направляются в обратном направлении не по пассивной, а по активной дуге пройденного дерева. Такие «лишние» ***запросы*** могут привести к генерации «лишних» движков, то есть движков, которые уничтожаются, не пройдя ни по одной непройденной ранее дуге. Заметим, что в исходном алгоритме в каждой нетерминальной вершине уничтожается ровно один такой «лишний» движок, а именно ждущий движок, если все дуги, выходяшие из вершины, уже законченные.

Возможны два решения этой проблемы.

1. «Лишние» сообщения ***запрос*** допускаются. Это может привести к дополнительному появлению «лишних» движков. Но число таких движков, очевидно, не превосходит числа нетерминальных вершин (как раньше) плюс число хорд, что не влияет на порядок верхней оценки сложности алгоритма.
2. «Лишние» сообщения ***запрос*** не допускаются. Регулятор узнаёт, как движок пришёл в вершину: по входящей дуге пройденного дерева (для начальной вершины – сгенерирован генератором) или по хорде, и в последнем случае не посылает сообщение ***запрос***. Для этого достаточно в параметрах сообщения ***куда идти*** указать признак того, что движок прошёл по хорде. Или ввести для этого случая новое сообщение ***куда идти после хорды*** с теми же параметрами, что и в сообщении ***куда идти***. Получив это новое сообщение, регулятор не посылает сообщение ***запрос***. В этом случае дополнительных «лишних» движков не появляется.

6.2. Петля

Когда движок проходит по непройденной дуге $a—i→b$, он должен опросить регуляторы, чтобы узнать, новая вершина b или уже пройденная. Однако, если $a=b$, то есть дуга $a—i→b$ является петлёй, то вершина b заведомо пройденная, а её регулятор – это регулятор вершины a. В этом случае опрос регуляторов

можно не проводить. Для того, чтобы узнать, является ли дуга $a—i→b$ петлёй, достаточно сравнить идентификаторы вершин a и b. Идентификатор вершины b движок узнаёт, пройдя по дуге $a—i→b$, то есть получив от графа сообщение ***ответ на проход***. Как движок узнает идентификатор вершины a?

При создании движка ему посылается ***ты движок***, в котором указывается как параметр идентификатор текущей вершины: начальной вершины, если движок создаётся генератором, или другой вершины, если движок создаётся регулятором (движком, ставшим регулятором новой вершины). Этот идентификатор вершины движок запоминает как *идентификатор текущей вершины*. После прохода по дуге движок получает сообщение ***ответ на проход***, в котором указывается идентификатор вершины, в которой движок оказался. Если это новая дуга, движок запоминает этот идентификатор как *идентификатор текущей вершины*. Однако, если это старая дуга, то в исходном алгоритме движок не запоминает идентификатор текущей вершины, поскольку ему известен адрес регулятора этой вершины.

Для оптимизации «петля» нужно следующее:

1. После прохода новой дуги движок сначала сравнивает идентификаторы в ячейке *идентификатор текущей вершины* и в параметрах сообщения ***ответ на проход***. Если они не совпали, движок запоминает новый идентификатор текущей вершины, дальнейшее поведение, как раньше. Если же идентификаторы совпали, то есть пройдена петля, движок выполняет те действия, которые он обычно выполняет после прохода по хорде и завершении опроса регуляторов.
2. После прохода старой дуги движок переписывает идентификатор вершины из параметров сообщения ***ответ на проход*** в ячейку *идентификатор текущей вершины*.

6.3. Кластеризация автоматов по машинам

Напомним, что под машиной мы понимаем процессор с памятью. Автомат работает в одной машине, но в одной машине может быть несколько автоматов. Передача сообщения между автоматами в одной и в разных машинах занимает существенно разное время, поэтому имеет смысл оптимизировать распределение автоматов по машинам.

Сравнивая нижнюю оценку $\boldsymbol{\Omega}(m)$ и верхнюю оценку $\boldsymbol{O}(m+nD)$, видим, что оптимизировать можно только опрос регуляторов, требующий $\boldsymbol{O}(n)$ тактов на каждый опрос и суммарно дающий $\boldsymbol{O}(nD)$ тактов.

Поскольку объём памяти машины $\boldsymbol{O}(1)$, в ней помещается $\boldsymbol{O}(1)$ автоматов. Если список регуляторов кластеризован, т.е. все регуляторы из одной машины расположены подряд, то та часть времени опроса, которая приходится на регуляторы одной машины равна $\boldsymbol{O}(1)$. Тем самым, время одного опроса равно $\boldsymbol{O}(p)$, где p — число машин. Это, конечно, не влияет на порядок оценки, поскольку $\boldsymbol{O}(n)=\boldsymbol{O}(p)$. Это равенство означает наличие такого коэфициента k,

что $n=kp+o(p)$, а на практике значение k очень важно, так как означает ускорение обхода в k раз.

Для кластеризации списка регуляторов нужно выделять память под регуляторы отдельно от памяти для движков и связанных с ними графов. Но в описанном выше алгоритме регулятор – это бывший движок. Поэтому приходится кластеризовать суммарно регуляторы и движки. Память под регуляторы и движки выделяется сначала в первой машине, а когда в ней нет места, то во второй машине, и так далее. В каждый момент времени имеется текущая машина, в которой эти автоматы создаются. Важно отметить, что освобождение памяти в машине из-под уничтожаемого движка не должно приводить к её использованию для новых движков и регуляторов, если это не текущая машина.

Но что делать, когда машины закончились, то есть текущая машина – последняя в списке машин и память в ней закончилась? Можно закончить работу, считая что граф слишком большой. Но можно продолжить обход с понижением скорости до тех пор, пока будет хватать памяти. Для этого будем использовать для создания новых движков и регуляторов свободную память уже не только в текущей машине. Опять сначала в первой машине, потом во второй машине и так далее по циклу. Скорость уменьшится из-за фрагментации по машинам списка регуляторов. Но мы всё же сможем обходить графы больших размеров.

6.4. Работа с ограниченной памятью

Что делать, если в процессе работы алгоритма памяти не хватает, то есть внешний автомат не может выполнить запрос на создание автомата или графа?

Если не может быть удовлетворён запрос на создание автомата (движка), поступивший от движка, ставшего регулятором, то можно вместо ответного сообщения ***автомат создан*** посылать сообщение ***нет памяти для нового автомата***. Получив такое сообщение движок, ставший регулятором, уничтожает связанный с ним граф, что приводит к появлению свободной памяти.

Если не может быть удовлетворён запрос на создание автомата (движка) или графа, поступивший от генератора, то такой запрос во внешнем автомате становится ждущим. Ждущий запрос может быть удовлетворён позже, когда появится свободная память из-под уничтожаемых графов.

Исключение составляет случай, когда уничтожается последний граф, но в этом случае никакая работа продолжена быть не может из-за нехватки памяти. Внешний автомат может выделить этот случай, «помня» число существующих графов и при уничтожении последнего графа сообщать о невозможности продолжать работу.

7. Заключение

Дальнейшие исследования обхода графов коллективом автоматов возможны по нескольким направлениям.

Первое направление: недетерминированные графы. Граф недетерминирован, если выходящая из вершины дуга идентифицируется неоднозначно: одному номеру выходящей дуги соответствует несколько дуг с общей начальной вершиной и разными конечными вершинами. Обход такого графа может пониматься двояко. 1) Полный обход графа, когда требуется проход по каждой дуге графа. 2) Частичный обход графа, когда нужно пройти хотя бы по одной дуге с данной начальной вершиной и данным номером выходящей дуги. Частичный обход графа применяется, когда есть уверенность, что интересующие нас свойства графа могут быть исследованы при таком обходе.

Для того, чтобы гарантировать возможность полного обхода недетерминированного графа, нужны какие-то дополнительные предположения. Одним из таких предположений может служить предположение об ограниченном недетерминизме, которое сводится к следующему: проходя t раз из данной вершины по выходящей дуге с данным номером мы пройдём все выходящие из этой вершины дуги с этим номером, где t заранее известная константа. Существуют алгоритмы такого обхода одним автоматом, оставляющем «пометки» в вершинах графа [[6]]. Можно поставить задачу разработки нового алгоритма такого обхода коллективом взаимодействующих автоматов (без «пометок» в вершинах).

Для того, чтобы гарантировать возможность частичного обхода недетерминированного графа, также нужны дополнительные предположения. Одним из таких предположений является, так называемая Δ-достижимость вершин графа из начальной вершины [[3]]. Говоря неформально, это означает, что существует алгоритм движения из начальной вершины в заданную вершину, который гарантированно обеспечивает достижение заданной вершины при любом недетерминированном выборе выходящей дуги в рамках множества дуг с тем же началом и тем же номером дуги.

Второе направление: использование внешней памяти. Предложенный в данной статье алгоритм обходит граф только в том случае, когда регуляторы всех нетерминальных вершин помещаются в суммарной памяти машин. Если это не так, становится актуальным вопрос об использовании дополнительной внешней памяти. Поскольку код программы регулятора, и вообще, всех автоматов: регуляторов, движков и генератора, один и тот же для всех этих автоматов, разные автоматы отличаются только состоянием и содержимым ячеек памяти. Поэтому речь идёт о размещении на внешней памяти этого набора данных. Адрес автомата, тем самым, может указывать на такой набор данных как в оперативной памяти той или иной машины, так и быть адресом по внешней памяти. Передача сообщения автомату, располагающемуся на

внешней памяти, потребует подкачки информации с внешней памяти, для чего, возможно, потребуется откачка части автоматов на внешнюю память.

Обход графа коллективом автоматов с использованием внешней памяти лежит на стыке двух направлений: 1) методы эффективной работы с внешней памятью, в частности, стратегий страничной и сегментной подкачки, 2) алгоритмы обхода графов коллективом автоматов. Сама постановка такой задачи является новой. Здесь потребуются новые алгоритмы (или модификации имеющихся алгоритмов) работы с внешней памятью, учитывающие специфику обхода графов, и алгоритмы обхода графов, учитывающие специфику работы с внешней памятью.

Третье направление: графы специального вида. Большой интерес представляет разработка новых алгоритмов исследования графов, принадлежащих тому или иному подклассу графов. Такие подклассы имеет смысл рассматривать при сочетании двух условий: эти подклассы имеют практическое значение (графы переходов исследуемых систем и сетей относятся к такому подклассу) и на этих подклассах алгоритмы могут работать быстрее.

Литература:

[1] И.Б. Бурдонов, А.С. Косачев, В.В. Кулямин, А.К. Петренко. "Подход UniTesK к разработке тестов" // Программирование, 2003 г., №6, с. 25-43.

[2] И.Б. Бурдонов, А.С. Косачев, В.В. Кулямин. "Неизбыточные алгоритмы обхода ориентированных графов. Детерминированный случай" // Программирование, 2003 г., №5, с. 59-69.

[3] И.Б. Бурдонов, А.С. Косачев, В.В. Кулямин. "Неизбыточные алгоритмы обхода ориентированных графов. Недетерминированный случай" // Программирование, 2004 г., №1, с. 2-17.

[4] И.Б. Бурдонов. "Обход неизвестного ориентированного графа конечным роботом" // Программирование, 2004 г., № 4, с. 11-34.

[5] И.Б. Бурдонов. "Проблема отката по дереву при обходе неизвестного ориентированного графа конечным роботом" // Программирование, 2004 г., № 6, с. 6-29.

[6] И. Бурдонов, А. Косачев. "Полное тестирование с открытым состоянием ограниченно недетерминированных систем". Программирование, 2009, №6, стр. 3-18.

[7] И.Б. Бурдонов, С.Г. Грошев, А.В. Демаков, А.С. Камкин, А.С. Косачев, А.А. Сортов. "Параллельное тестирование больших автоматных моделей" // Вестник ННГУ, 2011 г., №3, с. 187-193.

[8] A. Demakov, A. Kamkin, A. Sortov. "High-Performance Testing: Parallelizing Functional Tests for Computer Systems Using Distributed Graph Exploration". Open Cirrus Summit 2011, Moscow.

[9] И. Бурдонов, А. Косачев. "Обход неизвестного графа коллективом автоматов". Труды Международной суперкомпьютерной конференции "Научный сервис в сети Интернет: все грани параллелизма". 2013, изд. МГУ, стр. 228-232.

Graph learning by a set of automata

Igor Burdonov, Alexander Kosachev
ISP RAS, Moscow,Russia
{igor, kos}@ispras.ru

Abstract. Graph learning by automata is a basic task in many applications. Among these applications are verification and testing of software and hardware systems, network exploration including Internet and GRID basing on formal models. The system or network model, in the final analysis, is a transition graph with properties to be examined. In the recent years the size of the systems and networks in everyday use and, consequently, the size of their models and graphs to be learned grows constantly. Problems arise when graph exploration by a single automaton (computer) either requires a lot of time, or a lot of computer memory, or both. Therefore, there is a task of parallel and distributed graph exploration. This task is formalized as graph learning by a set of automata (a set of computers with sufficient total memory working in parallel). This paper covers the solution of this task. First of all the behavior of the automaton on the graph is formalized. Then the algorithm of the graph learning is described in detail. The estimation of the algorithm is $\boldsymbol{O}(m+nD)$. Some ways of possible optimization is discussed.

Keywords: graph learning, traversal of an unknown graph, finite automaton (state machine), extended finite state machine, communicating automata, parallel processing, distributed systems, testing.

References:

[1] Bourdonov I.B., Kossatchev A.S., Petrenko A.K., Kuliamin V.V. The UniTesK Approach to Designing Test Suites. Programming and Computer Software, Vol. 29, No. 6, 2003, pp. 310-322.

[2] Bourdonov I.B., Kossatchev A.S., Kuliamin V.V. Irredundant Algorithms for Traversing Directed Graphs: The Deterministic Case. Programming and Computer Software, Vol. 29, No. 5, 2003, pp. 245-258.

[3] Bourdonov I.B., Kossatchev A.S., Kuliamin V.V. Irredundant Algorithms for Traversing Directed Graphs: The Nondeterministic Case. Programming and Computer Software, Vol. 30, No. 1, 2004, pp. 2-17.

[4] Bourdonov I.B. Traversal of an Unknown Directed Graph by a Finite Robot. Programming and Computer Software, Vol. 30, No. 4, 2004, pp. 188-203.

[5] Bourdonov I.B. Backtracking Problem in the Traversal of an Unknown Directed Graph by a Finite Robot. Programming and Computer Software, Vol. 30, No. 6, 2004, pp. 305-322.

[6] Bourdonov I.B., Kossatchev A.S. Complete Open□State Testing of Limitedly Nondeterministic Systems. Programming and Computer Software, Vol. 35, No. 6, 2009,pp.301-313

[7] Bourdonov I.B., Groshev S.G., Demakov A.V., Kamkin A.S., Kossatchev A.S, Sortov A.A. Parallel'noe testirovanie bol'shikh avtomatnykh modelej [Parallel testing of

large automata models], Vestnik NNGU [Vestnik of UNN], №3920, 2011, pp. 187-193. (in Russian)

[8] A. Demakov, A. Kamkin, A. Sortov. "High-Performance Testing: Parallelizing Functional Tests for Computer Systems Using Distributed Graph Exploration". Open Cirrus Summit 2011, Moscow.

[9] Bourdonov I.B., Kossatchev A.S. Obkhod neizvestnogo grafa kollektivom avtomatov [Unknown graph traversing by automata group]. Trudy Mezhdunarodnoj superkomp'yuternoj konferentsii “Nauchnyj servis v seti Internet: vse grani parallelizma ”(21-26 sentyabrya 2009 g., g. Novorossijsk) [The proceeding of Russian Supercomputer conference ‘Scientific service of Internet’ (2013, Novorossijsk)] – Moscow, MSU publ., 2013, pp. 228-232. (in Russian)