# Обход неизвестного графа коллективом автоматов. Недетерминированный случай

*Игорь Бурдонов <igor@ispras.ru>,*
*Александр Косачев <kos@ispras.ru>*
*Институт системного программирования РАН,*
*109004, Россия, г. Москва, ул. А. Солженицына, д. 25.*

**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 ориентированному графу переходов, свойства которого нужно исследовать. За последние годы размер реально используемых систем и сетей и, следовательно, размер их графовых моделей 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. В основе такого исследования лежит обход графа (проход по всем его дугам, достижимым из начальной вершины). Автоматы могут генерироваться в начальной вершине графа, перемещаться по дугам графа в направлении их ориентации и обмениваться между собой сообщениями через независимую (от графа) сеть связи. Суммарная память автоматов используется для хранения описания пройденной части графа. Для перемещения из вершины по выходящей из неё дуге графа автомат каким-то образом должен идентифицировать эту дугу: указать её номер. В нашей статье «Обход неизвестного графа коллективом автоматов» предложен алгоритм такого обхода для случая детерминированных графов. Задача усложняется, если граф недетерминирован. В таком графе одному номеру дуги соответствует, вообще говоря, несколько дуг, из которых для перехода выбирается одна дуга недетерминированным образом. Для того, чтобы обход графа был возможен, должна быть гарантия, что при неограниченном числе экспериментов каждая выходящая из вершины дуга с данным номером может быть пройдена. Такой недетерминизм мы называем справедливым. Решению задачи обхода справедливо недетерминированных графов посвящена данная работа.

**Ключевые слова:** недетерминированные графы, исследование графа, обход графа, взаимодействующие автоматы, параллельная обработка, распределенные системы, тестирование.





**DOI:** 10.15514/ISPRAS-2015-27(1)-4

**Для цитирования:** Бурдонов Игорь, Косачев Александр. Обход неизвестного графа коллективом автоматов. Недетерминированный случай. Труды ИСП РАН, том 27, вып. 1, 2015 г., стр. 51-68. DOI: 10.15514/ISPRAS-2015-27(1)-4.

## *1. Введение*

Задача обхода неизвестного ориентированного графа автоматом (прохода автоматом по всем дугам графа, достижимым из начальной вершины) используется во многих приложениях. В данной статье подразумевается тестирование, когда граф — это граф автомата тестируемой системы, автомат на графе — тестирующая система, а проход по дуге — это тестовое воздействие и наблюдение результата [[4]]. В качестве практического примера можно привести работу [[11]], 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.

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. Алгоритм обхода графа одним автоматом предложен в [[5]]. Если число автоматов больше одного, но ограничено, то распараллеливание ускоряет обход, но не меняет порядок времени обхода в наихудшем случае [[10]].

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: автоматы не могут обмениваться между собой сообщениями, но могут писать/читать из текущей вершины символы конечного алфавита и перемещаться по дугам графа. Это эквивалентно «обратной» модели, когда автоматы неподвижно «сидят» в вершинах графа, а по дугам передаются сообщения, которые автоматы посылают друг другу. Такой подход может применяться, например, для сети интернета, когда вершина — это узел сети, а проход по дуге — передача сообщения между узлами. Алгоритм работы одного конечного автомата

предлагается в [[3],[7],[8]]. Если конечных автоматов несколько, это уменьшает время обхода [[15]].

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 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. Именно этот подход для конечных графов рассматривается в настоящей статье.

Когда автомат создаётся, ему выделяется отдельный клон (экземпляр) исследуемого графа. Считается, что автомат находится в выделенной начальной вершине графа, которую мы будем называть *корнем*. Этому соответствует рестарт тестируемой системы, т.е. её перевод в начальное состояние. Суммарная память автоматов используется для хранения описания пройденной части графа. После создания автоматы могут перемещаться по дугам графа в направлении их ориентации (каждый автомат по своему клону графа) и обмениваться между собой сообщениями через независимую (от графа) сеть связи.

Для перемещения из вершины по выходящей из неё дуге графа автомат каким-то образом должен идентифицировать эту дугу: указать её номер. Этому соответствует то или иное тестовое воздействие на тестируемую систему. Предполагается, что все дуги, выходящие из вершины перенумерованы, начиная с 1, и известно число выходящих дуг (число допустимых тестовых воздействий). После прохода по дуге автомат должен каким-то образом узнать, в какой вершине он оказался, и сколько у неё выходящих дуг. Для этого используется уникальный идентификатор вершины. Этому соответствует тестирование с открытым состоянием [[9]], когда тест после тестового воздействия может узнать текущее состояние реализации с помощью примитива *status message*. Взаимодействие автомата с графом сводится к примитиву ***проход по дуге*** с параметром «номер дуги» и ***ответа на проход***, в котором сообщается идентификатор конца дуги. Кроме того, автомат должен знать число дуг, выходящих из вершины. При тестировании это число допустимых тестовых воздействий, которое может зависеть от предыистории взимодействия. В графовой модели для простоты мы будем считать, что это число сообщается автомату после прохода по дуге вместе с идентификатором конца дуги. Мы будем считать, что имеется некий автомат графа, а указанный примитив и ответ на него реализуются как сообщения, посылаемые из автомата-обходчика автомату графа и обратно.

При обмене сообщениями автомат-получатель (включая автомат графа) идентифицируется адресом автомата. Автомат получает адрес при его создании, и этот адрес является параметром ответа на сообщение ***создай автомат***. Сами автоматы и клоны графа создаются и уничтожаются неким неописываемым внешним автоматом. Он же инициирует обход,





создавая первый автомат, и ему же посылается сообщение о завершении обхода.

Этот подход впервые был применён в нашей работе [[12]], а в [[13],[14]] описана улучшенная модификация алгоритма обхода. В этих работах граф предполагается детерминированным. Это означает, что в каждой вершине номер дуги однозначно определяет выходящую из этой вершины дугу. Когда автомат в текущей вершине указывает номер дуги как параметр примитива ***проход по дуге***, он может перейти по дуге только в одну вершину – конец дуги с указанным номером. Если это не так, граф считается недетерминированным. В данной работе мы предлагаем алгоритм обхода такого графа при некоторых предположениях о характере его недетерминизма.

## 2. Недетерминизм

Граф с нумерацией дуг, выходящих из каждой вершины, недетерминирован, если одному номеру дуги может соответствовать несколько дуг. Множество дуг с общим началом и одним номером дуги будем называть *дельта-дугой*. Будем считать, что недетерминированный выбор дуги по дельта-дуге осуществляется с помощью недетерминированной функции выбора $s$ : *«множество дельта-дуг»* $\to$ *«множество дуг»*, которая для каждой дельта-дуги $X$ недетерминированным образом возвращает дугу $x \in X$. Автомат указывает в текущей вершине номер дуги, который однозначно определяет дельту-дугу, и проходит некоторую дугу из дельта-дуги, определяемую функцией выбора. Мы будем говорить также, что автомат проходит дельта-дугу.

Поскольку после прохода по дуге автомат узнаёт только идентификатор конца дуги, кратные дуги с одним номером неразличимы между собой. Тем самым, без дополнительных предположений невозможно гарантировать проход по всем дугам графа. Мы будем считать, что в графе нет кратных дуг с одним номером. Тогда любая дуга уникально идентифицируется идентификатором её начала, номером дуги и идентификатором конца дуги.

Для каждой дельта-дуги $X$ будем рассматривать различные последовательности значений функции выбора $s(X), s(X), \ldots$. Интуитивно ясно, что если может быть получена некоторая такая последовательность, то может быть получен и любой её префикс.

Будем говорить, что граф *абсолютно недетерминирован*, если для любой дельта-дуги $X$ любая бесконечная последовательность дуг из этой дельта-дуги $S \in X^\infty$ может быть получена как последовательность значений функции выбора $S = s(X), s(X), \ldots$. Это означает, что если на недетерминизм функции выбора не налагается никаких ограничений, обход недетерминированного графа невозможно выполнить за конечное время, поскольку нет никаких гарантий, что за конечное число проходов по дельта-дуге удастся пройти по всем её дугам, если их больше одной. Например, всё время может выбираться одна и та же дуга.

Будем говорить, что граф *справедливо недетерминирован*, если для любой дельта-дуги $X$ в любой бесконечной последовательности значений функции выбора $S = s(X), s(X), \ldots$ каждая дуга $x \in X$ встречается бесконечное число раз. Это означает, что для прохода по всем дугам дельта-дуги достаточно конечного числа проходов по дельта-дуге. Поэтому, если известно число дуг в дельта-дуге, автомат может за конечное время не только пройти по всем дугам дельта-дуги, но и узнать об этом. А, кроме того, по одной и той же дуге можно проходить любое неограниченное число раз, просто повторяя достаточно большое число раз проход по дельта-дуге.

Можно отметить, что гипотеза о справедливом недетерминизме графа, фактически, эквивалентна гипотезе о глобальном тестировании [[1],[2]], которая предполагает, что при бесконечном числе прогонов теста будут получены все возможные варианты поведения тестируемой системы. Для графа такому варианту поведения системы соответствует маршрут в графе, а для того, чтобы можно было пройти любой маршрут, функция выбора должна быть справедливой.

Частным случаем справедливо недетерминированного графа является *ограниченно недетерминированный граф*. Это такой граф, для которого существует такое число $t$, что для любой дельта-дуги $X$ и любой бесконечной последовательности значений функции выбора $S = s(X), s(X), \ldots$ каждая дуга $x \in X$ встречается хотя бы один раз на каждом отрезке последовательности длиной $t$, т.е. на отрезке $S[i..i+t-1]$ для каждого натурального числа $i$. Для данного числа $t$ такой граф будем называть также *$t$-недетерминированным*.

Очевидно, что при $t=1$ мы имеем детерминированный граф.

Имеет место следующая последовательность строгих вложений классов графов:

класс детерминированных графов

$\subset$ класс $t$-недетерминированных графов

$\subset$ класс ограниченно недетерминированных графов

$\subset$ класс справедливо недетерминированных графов

$\subset$ класс абсолютно недетерминированных графов.

В следующем разделе будет предложен алгоритм обхода коллективом автоматов любого справедливо недетерминированного конечного графа, для которого 1) в дельта-дуге нет кратных дуг, 2) известно число дельта-дуг, выходящих из каждой вершины, 3) известно число дуг в каждой дельта-дуге. Кроме того, мы будем предполагать, что при передаче сообщения «от точки к точке» не происходит потери, искажения, генерации лишних или обгона сообщений.

Замечание: Для $t$-недетерминированного графа можно ослабить требования, оставив только знание числа дельта-дуг в каждой вершине. Действительно, если мы $t$ раз выполним проход по дельта-дуге $X$, каждый раз запоминая дугу, по которой мы проходим, то, во-первых, мы гарантированно пройдём по всем





дугам дельта-дуги, включая кратные дуги, а, во-вторых, узнаем число дуг в дельта-дуге с точностью до кратности, т.е. узнаем число различных концов дуг этой дельта-дуги.

## *3. Алгоритм обхода*

Алгоритм основан на трёх режимах работы автоматов: *генератор*, *регулятор* и *движок*. Обход графа, то есть движение по его дугам, выполняют движки, регуляторы предназначены для управления перемещением движков по дугам, а генератор создаёт и уничтожает (с помощью внешнего автомата) движки и связанные с ними клоны графов. С каждой нетерминальной (имеющей выходящие дуги) вершиной связан один регулятор, в котором хранится описание дельта-дуг и дуг, выходящих из вершины. Регулятор корня выполняет также функции генератора. Для управления перемещением движков регулятор сообщает движку, находящемуся в вершине регулятора, номер дуги, однозначно определяющий дельту-дугу. Движок выполняет проход по этой дельта-дуге, что означает проход по одной из дуг этой дельта-дуги, выбираемой функцией выбора.

### 3.1 Описание дуги

В процессе работы алгоритма в суммарной памяти регуляторов строится описание пройденной части графа. Для этого в регуляторе вершины создаётся описание каждой выходящей из вершины дуги. *Пройденные* дуги делятся на *прямые* и *хорды*. Прямые дуги образуют остов пройденной части графа. *Законченной* дугой будем называть дугу, по которой больше не нужно посылать движки. *Непройденная* дуга всегда незаконченная, хорда всегда законченная, а прямая дуга может быть сначала незаконченной, а потом стать законченной. Прямая дуга становится законченной, когда выше неё по остову нет вершин, из которых выходят непройденные дуги. Заметим, что, поскольку регулятор посылает движки по дельта-дуге, а не отдельной дуге, движок всё равно может пройти по законченной дуге, но только в том случае, если в дельта-дуге есть незаконченная дуга. В целом имеются следующие состояния дуг: непройденная, хорда, незаконченная прямая и законченная прямая. В конце обхода все дуги законченные, т.е. хорды и законченные прямые.

Вершину будем называть *законченной*, если все выходящие из неё дуги законченные.

Описание дуги состоит из следующих полей:

- состояние дуги;
- идентификатор конца дуги [несущественен для непройденной дуги];
- адрес регулятора конца дуги [несущественен для непройденной дуги].

## 3.2 Сообщения

Сообщение состоит из названия (тега) и параметров сообщения. Ниже перечислены все используемые сообщения и их параметры, кроме адреса получателя, который является обязательным параметром каждого сообщения. В скобках указаны отправитель и получатель сообщения.

1. ***ты генератор*** (внешний автомат → первый созданный автомат):
   - адрес внешнего автомата,
   - адрес генератора, то есть адрес получателя сообщения;
2. ***создай граф*** (генератор → внешний автомат):
   - адрес генератора, то есть адрес отправителя сообщения;
3. ***граф создан*** (ответ на ***создай граф***):
   - адрес клона графа,
   - идентификатор корня,
   - число выходящих из корня дельта-дуг;
   - список чисел дуг для каждой дельта-дуги в корне;
4. ***создай автомат*** (генератор или движок, ставший регулятором, → внешний автомат):
   - адрес отправителя сообщения;
5. ***автомат создан*** (ответ на ***создай автомат***):
   - адрес созданного автомата;
6. ***ты движок*** (создатель движка → созданный движок):
   - адрес движка, т.е. адрес получателя сообщения,
   - адрес регулятора корня (он же генератор),
   - адрес клона графа,
   - идентификатор текущей вершины,
   - адрес регулятора текущей вершины;
7. ***куда идти*** (движок → регулятор текущей вершины):
   - адрес движка, т.е. адрес отправителя сообщения;
8. ***иди по дуге*** (ответ на ***куда идти***):
   - номер выходящей дуги (может быть равным нулю);
9. ***проход по дуге*** (движок → клон ассоциированного с движком графа):
   - адрес движка, т.е. адрес отправителя сообщения,
   - номер выходящей дуги;
10. ***ответ на проход*** (ответ на ***проход по дуге***):
    - идентификатор вершины конца пройденной дуги,
    - число выходящих из этой вершины дельта-дуг,
    - список чисел дуг для каждой дельта-дуги;
11. ***извещение*** (движок, прошедший по дуге, → регулятор начала дуги):
    - номер пройденной дуги,
    - описание пройденной дуги;





12. ***опрос*** (движок → регулятор корня как первый в списке регуляторов, или регулятор → следующий регулятор в списке регуляторов):
    - адрес движка, инициировавшего опрос,
    - идентификатор вершины;
13. ***ответ на опрос*** (найденный регулятор → движок, инициировавший опрос):
    - адрес найденного регулятора,
    - номер прямой дуги, входящей в вершину найденного регулятора;
    - адрес регулятора начала прямой дуги, входящей в вершину найденного регулятора;
14. ***уничтожь граф*** (движок перед самоуничтожением → внешний автомат):
    - адрес клона графа;
15. ***уничтожь автомат*** (самоуничтожающийся движок → внешний автомат):
    - адрес самоуничтожающегося движка, т.е. адрес отправителя.

## 3.3 Состояние автомата

Состояние автомата – это состояние памяти, в которой имеются следующие поля:

- режим работы: генератор (он же регулятор корня), регулятор, движок;
- собственный адрес автомата;
- адрес внешнего автомата;
- адрес клона графа;
- адрес генератора;
- адрес следующего в списке регуляторов (у последнего в списке регулятора адрес пустой);
- идентификатор текущей вершины;
- адрес регулятора текущей вершины;
- номер входящей дуги;
- адрес регулятора начала входящей дуги;
- число выходящих дельта-дуг;
- список по выходящим дельта-дугам (от 1 до числа выходящих дельта-дуг):
  - число дуг в дельта-дуге;
  - список описаний дуг дельта-дуги (от 1 до числа дуг в дельта-дуге).

## 3.4 Описание алгоритма

### 3.4.1 Генератор

Работа алгоритма начинается, когда извне создаётся первый автомат и ему посылается сообщение ***ты генератор***. Генератор выполняет функции регулятора корня, описанные ниже, и, кроме того, непрерывно генерирует

движки. Сначала создаётся клон графа с помощью сообщений ***создай граф*** и ответного сообщения ***граф создан***. Затем создаётся автомат с помощью сообщений ***создай автомат*** и ответного сообщения ***автомат создан***. Созданному автомату посылается сообщение ***ты движок*** с передачей ему адреса клона графа. Генерация движков происходит до тех пор, пока корень не станет законченным. Когда это произойдёт, генератор посылает вовне ***извещение***, означающее конец обхода.

### 3.4.2 Регулятор

Регулятор связан с одной вершиной графа и регулирует движение движков, попадающих в эту вершину, по выходящим дугам. Для этого регулятор ожидает от приходящих движков сообщений ***куда идти***. Получив такое сообщение, регулятор проверяет, закончена ли вершина. Если есть незаконченная дуга, регулятор посылает движку её номер в ответном сообщении ***иди по дуге***. Если вершина закончена, регулятор посылает движку нулевой номер дуги в ответном сообщении ***иди по дуге***.

Также регулятор может получить сообщение ***извещение***, содержащее описание дуги, выходящей из вершины регулятора. Регулятор проверяет по номеру дуги и идентификатору конца дуги есть ли уже описание этой дуги или ещё нет. Если такой дуги ещё нет, регулятор создаёт её описание на месте описания одной из непройденных дуг. Если такая дуга уже есть, регулятор меняет её состояние на то, которое указано в ***извещении***, за исключением случая, когда в ***извещении*** указана незаконченная прямая дуга, а в регуляторе дуга уже законченная.

Это исключение может возникнуть из-за асинхронности поведения автоматов. Если по непройденной дуге, ведущей в непройденную вершину, проходит первый движок, он посылает ***извещение*** №1 о незаконченной прямой дуге. Однако это ***извещение*** №1 может идти достаточно долго, и за это время по дуге может пройти много движков. В конце конйов один из этих движков пошлёт ***извещение*** №2 о том, что дуга законченная, и это ***извещение*** №2 обгонит ***извещение*** №1.

### 3.4.3 Движок

Движок начинает свою работу с получения сообщения ***ты движок***. Движок спрашивает у регулятора текущей вершины ***куда идти*** и ожидает ответа ***иди по дуге***.

Если в ответном сообщении ***иди по дуге*** указан нулевой номер дуги, движок должен уничтожить граф и самого себя с помощью сообщений ***уничтожь граф*** и ***уничтожь автомат***.

Если в ответном сообщении ***иди по дуге*** указан ненулевой номер дуги, движок выполняет проход по дуге.

Мы опишем цикл работы движка, начиная с прохода по дуге.





Для прохода по дуге движок посылает клону графу сообщение ***проход по дуге*** и ожидает ответа. В сообщении ***ответ на проход***, движок получает идентификатор текущей вершины (конца пройденной дуги). По этому идентификатору движок должен найти регулятор текущей вершины или убедиться, что такого регулятора нет. Для этого движок выполняет опрос регуляторов.

Опрос начинается с того, что движок посылает сообщение ***опрос*** регулятору корня, который всегда находится в начале списка регуляторов. Каждый регулятор, получив такое сообщение, сравнивает идентификатор вершины из сообщения с идентификатором вершины, регулятором которой он является. Если они совпадают, регулятор отправляет движку, инициировавшему опрос, сообщение ***ответ на опрос***. В противном случае регулятор пересылает сообщение ***опрос*** дальше по списку регуляторов. При этом последний в списке регулятор (у него пустой адрес следующего в списке регуляторов) становится предпоследним, запоминая адрес движка, инициировавшего опрос, как адрес следующего в списке регуляторов и пересылая ***опрос*** этому движку.

1. Если регулятор текущей вершины не найден (движок получил сообщение ***опрос***), движок сам становится регулятором текущей вершины и посылает регулятору начала пройденной дуги ***извещение*** о незаконченной прямой дуге. Затем движок, ставший регулятором, создаёт новый движок с помощью сообщения ***создай автомат*** и ответного сообщения ***автомат создан*** и посылает созданному автомату сообщение ***ты движок***, в котором передаёт ему адрес клона графа.
2. Если регулятор текущей вершины найден (движок получил сообщение ***ответ на опрос***), движок узнаёт адрес регулятора текущей вершины, а также номер прямой дуги, входящей в вершину найденного регулятора, и адрес регулятора начала прямой дуги, входящей в вершину найденного регулятора. Если найден регулятор корня, то у него номер входящей прямой дуги равен нулю. Движок определяет, в какое состояние должна перейти пройденная им дуга: хорда или прямая дуга.

   Здесь нужно отметить, что эта дуга была отмечена как непройденная в тот момент, когда движок получил номер этой дуги от регулятора её начала в сообщении ***иди по дуге***. Однако после этого движок выполнял проход по дуге и опрос регуляторов. Из-за асинхронности поведения автоматов могло оказаться, что по этой дуге уже прошёл другой движок, попал в ещё непройденную вершину, стал её регулятором, послал ***извещение*** о прямой дуге, но это ***извещение*** не поступило в регулятор начала дуги к указанному выше моменту. В этом случае наш движок не должен посылать ***извещение*** о хорде, а повторное ***извещение*** о прямой дуге может понадобиться только в том случае, если она должна быть отмечена как законченная. Если же эта дуга хорда, и предыдущий движок, прошедший по той же дуге, уже послал ***извещение*** об этом, то повторное ***извещение*** о хорде ничего не изменит в описании этой дуги, поэтому его

можно послать. Если же наш движок был первым, прошедшим по этой дуге, то такое ***извещение*** о хорде нужно посылать обязательно.

Как движок определяет состояние пройденной дуги? Это прямая дуга, если номер пройденной дуги и адрес регулятора её начала совпадают с номером прямой дуги, входящей в вершину найденного регулятора, и адресом регулятора начала прямой дуги, входящей в вершину найденного регулятора. В противном случае это хорда.

2.1. Если движок прошёл по хорде, он посылает регулятору её начала ***извещение*** о хорде. После этого движок спрашивает у регулятора текущей вершины ***куда идти*** и ожидает ответа ***иди по дуге***. Если в ответном сообщении ***иди по дуге*** указан нулевой номер дуги, движок уничтожает граф и самого себя с помощью сообщений ***уничтожь граф*** и ***уничтожь автомат***. Если в ответном сообщении ***иди по дуге*** указан ненулевой номер дуги, движок выполняет проход по дуге.

2.2. Если движок прошёл прямую дугу, он спрашивает у регулятора текущей вершины ***куда идти*** и ожидает ответа ***иди по дуге***. Если в ответном сообщении ***иди по дуге*** указан нулевой номер дуги, движок сначала посылает регулятору начала пройденной дуги ***извещение*** о законченной прямой дуге, а потом уничтожает граф и самого себя с помощью сообщений ***уничтожь граф*** и ***уничтожь автомат***. Если в ответном сообщении ***иди по дуге*** указан ненулевой номер дуги, движок выполняет проход по дуге, и никакого ***извещения*** не посылает.

## 3.5 Оптимизация

В этом разделе мы опишем оптимизации, позволяющие ускорить работу алгоритма.

### 3.5.1 Терминальный корень

Если корень – терминальная вершина, то генератору не нужно создавать движок. После создания первого клона графа генератор проверяет, является ли корень терминальной вершиной. Если да, то генератор уничтожает клон графа с помощью сообщения ***уничтожь граф*** и посылает вовне ***извещение***, означающее конец обхода.

### 3.5.2 Терминальная некорневая вершина

Аналогично, если движок прошёл по дуге в терминальную вершину, и это первый движок, попавший в эту вершину, то после того, как он становится регулятором, ему не нужно создавать новый движок. В этом случае движок, ставший регулятором, посылает регулятору начала пройденной дуги





***извещение*** о законченной прямой дуге. Затем движок, ставший регулятором, уничтожает граф с помощью сообщения ***уничтожь граф***.

### 3.5.3 Петля

Пройденная петля по определению является хордой. Движок может распознать этот случай без опроса регуляторов. Для этого после прохода дуги достаточно сравнить идентификатор текущей вершины с идентификатором той вершины, из которой движок вышел. Если они совпадут, то движок прошёл по петле.

### 3.5.4 Пройденная дуга

Если движок проходит по уже пройденной дуге, он может определить её конец без опроса регуляторов. Для этого достаточно воспользоваться описанием пройденных дуг с тем же номером, которое имеется в регуляторе начала дуги. Это описание регулятор может передать движку как дополнительный параметр сообщения ***иди по дуге***: список описаний пройденных дуг с данным номером дуги.

Сначала по идентификатору текущей вершины движок проверяет, прошёл он уже пройденную дугу или ещё не пройденную. Для этого он сравнивает идентификатор текущей вершины с идентификаторами концов пройденных дуг пройденной дельта-дуги. Если движок прошёл непройденную дугу, его поведение описано в основном алгоритме. Если движок прошёл пройденную дугу, его работа зависит от состояния этой дуги.

1. Если движок прошёл пройденную хорду, движок знает также адрес регулятора текущей вершины, спрашивает у него ***куда идти*** и ожидает ответа ***иди по дуге***. Если в ответном сообщении ***иди по дуге*** указан нулевой номер дуги, движок уничтожает граф и самого себя с помощью сообщений ***уничтожь граф*** и ***уничтожь автомат***. Если в ответном сообщении ***иди по дуге*** указан ненулевой номер дуги, движок выполняет проход по дуге.
2. Если движок прошёл пройденную незаконченную прямую дугу, движок знает также адрес регулятора текущей вершины, спрашивает у него ***куда идти*** и ожидает ответа ***иди по дуге***. Если в ответном сообщении ***иди по дуге*** указан нулевой номер дуги, движок сначала посылает регулятору начала пройденной дуги ***извещение*** о законченной прямой дуге, а потом уничтожает граф и самого себя с помощью сообщений ***уничтожь граф*** и ***уничтожь автомат***. Если в ответном сообщении ***иди по дуге*** указан ненулевой номер дуги, движок выполняет проход по дуге.
3. Если движок прошёл пройденную законченную прямую дугу, движок уничтожает граф и самого себя с помощью сообщений ***уничтожь граф*** и ***уничтожь автомат***.

## 3.6 Теорема о конце обхода

Будем предполагать, что время передачи одного сообщения, время срабатывания каждого автомата и время перемещения автомата по дуге (т.е. время срабатывания автомата клона графа) ограничено.

Теорема о конце обхода: Через конечное время после начала работы обход графа будет завершён, то есть по каждой дуге графа хотя бы один раз прошёл хотя бы один движок, генератор пошлёт внешнему автомату ***извещение***. Кроме того, все дуги будут законченными, а прямые дуги будут образовывать остов графа, ориентированный от корня.

Доказательство:

Из описания алгоритма следует, что состояние дуги может меняться так, как это изображено на **Ошибка! Источник ссылки не найден.**.

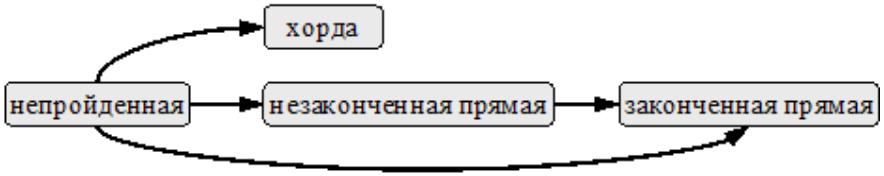


*Рис. 1. Схема изменения состояния дуги*

Поскольку в графе изменения состояний дуги нет ориентированных циклов, а число дуг конечно, начиная с некоторого времени $T_1$ все дуги перестают менять своё состояние. Далее будем рассматривать работу алгоритма после времени $T_1$.

Дуга *ab* становится прямой, если она была непройденной и вела в непройденную вершину *b*, после того, как по этой дуге прошёл движок, оказавшийся первым среди всех движков попадающих в вершину *b*. Поэтому прямые дуги не образуют ориентированных циклов и в каждую вершину входит не более одной прямой дуги. Также, поскольку движки начинают двигаться по графу, начиная с корня, все пройденные вершины достижимы из корня по пройденным дугам, и в каждую пройденную вершину, кроме корня, входит прямая дуга. Отсюда следует, что после времени $T_1$ прямые дуги образуют остов пройденной части графа, а хорды являются хордами этого остова, т.е. начинаются и заканчиваются в вершинах остова, но сами остову не принадлежат.

Покажем, что через конечное время генератор прекратит генерацию движков.

Допустим, это не так и движки генерируются бесконечно долго. Поскольку число путей в графе конечно, существуют пути, по которому движки двигаются бесконечное число раз. Как их префиксы существуют пути из прямых незаконченных дуг, начинающиеся в корне, по которым движки двигаются бесконечное число раз. Возьмём





максимальный из таких путей путь *P*. Покажем, что каждый движок, проходящей путь *P*, должен двигаться дальше из конца пути.

Допустим, это не так. Движок не двигается дальше, если конец пути законченный. Если путь *P* имеет нулевую длину, то его конец совпадает с его началом – корнем, а тогда корень законченный, и генератор должен был бы прекратить генерацию движков, что противоречит допущению. Если путь *P* имеет ненулевую длину, то через конечное время последняя дуга пути *P* станет законченной, что противоречит тому, что после времени $T_1$ все дуги перестают менять своё состояние. Мы пришли к противоречию, следовательно, наше допущение не верно, и утверждение доказано.

Поскольку число дульта-дуг с началом в конце пути *P* конечно, по одной из этих дельта-дуг движки двигаются бесконечно долго. Но тогда в этой дельта-дуге есть незаконченная дуга. В силу справедливости недетерминизма и конечности числа дуг в дельта-дуге по этой дуге также бесконечно долго двигаются движки. Эта дуга не может быть прямой, так как это противоречило бы максимальности пути *P*. Также эта дуга не может быть непройденной, так как после первого же прохода по этой дуге через конечное время она изменит своё состояние, что противоречит тому, что после времени $T_1$ все дуги перестают менять своё состояние. Однако других незаконченных дуг не бывает. Мы пришли к противоречию, следовательно, наше допущение не верно, и утверждение доказано.

Итак, мы доказали, что через конечное время генератор прекратит генерацию движков. Покажем, что в этот момент времени все дуги графа пройдены и закончены, т.е. помечены в регуляторах их начал как хорды или законченные прямые дуги, и, следовательно, прямые дуги образуют остов графа.

Допустим, это не так: существует незаконченная дуга. Поскольку все вершины достижимы из корня, должна существовать незаконченная дуга *ab*, начало которой, вершина *a*, принадлежит остову пройденной части графа. Тогда существует прямой путь *Q* из корня в вершину *a*. Вершина, из которой выходит незаконченная дуга, по определению незаконченная. Если вершина незаконченная, то входящая в неё прямая дуга также незаконченная. Отсюда следует что весь путь *Q* незаконченный: все его дуги и вершины. В частности, не закончено начало пути – корень, но это противоречит тому, что генератор прекратил генерацию движков. Мы пришли к противоречию, следовательно, наше допущение не верно, и утверждение доказано.

Теорема доказана.

## 4. Заключение

Дальнейшие исследования обхода недетерминированных графов коллективом автоматов возможны по нескольким направлениям.

Одно направление – это то или иное уточнение справедливой функции выбора, которое позволило бы оценить время работы алгоритма. Одним из таких уточнений является *t*-недетерминизм. В [[14]] для детерминированного случая ($t=1$) доказана оценка $\boldsymbol{O}(m+nD)$, где $m$ – число дуг в исходном графе, $n$ – число вершин, $D$ – диаметр графа (длина максимального пути). В [[9]] для неавтоматного обхода (что эквивалентно обходу одним автоматом, в память которого помещается весь граф) *t*-недетерминированных графов доказана оценка $\boldsymbol{O}(bt^n)$, где $b$ – ограничение на число дельта-дуг, выходящих из одной вершины. Можно предположить, что для коллектива автоматов оценка останется экспоненциальной. Это определяется тем, что для прохода пути длиной $k$ мы должны $k$ раз применить функцию выбора и каждый раз получить требуемую дугу, следовательно, для гарантированного прохода этого пути нам может понадобиться $t^k$ попыток. Было бы интересно рассмотреть другие функции выбора, для которых оценка меньше экспоненциальной. В качестве простого примера можно рассмотреть граф с числом недетерминированных дуг, ограниченным константой $k$. В этом случае экспоненциальная составляющая времени обхода не превысит $t^k$.

Другое направление – снижение требований к обходу. Можно потребовать прохода не по каждой дуге, а по каждой дельта-дуге графа. В терминах тестирования это означает, что нужно попробовать каждое тестовое воздействие в каждом состоянии системы, но необязательно получать все возможные варианты поведения системы на эти тестовые воздействия. Такой обход в [[6]] называется Δ-обходом. Доказаны необходимые и достаточные условия существования Δ-обхода, которые базируются на понятии Δ-достижимости вершин. Говоря неформально, из вершины $a$ Δ-достижима вершины $b$, если существует алгоритм движения из вершины $a$ в вершину $b$, который гарантированно обеспечивает достижение вершины $b$ при любой (даже не справедливой) функции недетерминированного выбора. Для того, чтобы Δ-обход можно было было выполнить без рестарта граф должен быть сильно-Δ-связным, т.е. каждая вершина должна быть Δ-достижима из каждой вершины. При наличии рестарта достаточно Δ-достижимости каждой вершины из корня графа, в этом случае Δ-обход назван Δ-покрытием. Неавтоматный Δ-обход можно выполнить за время $\boldsymbol{O}(nm)$. Было бы интересно попытаться модифицировать этот алгоритм для коллектива взаимодействующих автоматов.





## Литература:


[1]. R. Milner. "A modal characterisation of observable machine-behaviour". In G. Astesiano & C. Bohm, editors: Proceedings CAAP 81, LNCS 112, Springer-Verlag, 1981, pp. 25-34.

[2]. van Glabbeek R.J. The linear time - branching time spectrum II; the semantics of sequential processes with silent moves. Proceedings CONCUR '93, Hildesheim, Germany, August 1993 (E. Best, ed.), LNCS 715, Springer-Verlag, 1993, pp. 66-81.

[3]. Y. Afek and E. Gafni, Distributed Algorithms for Unidirectional Networks, SIAM J. Comput., Vol. 23, No. 6, 1994, pp. 1152-1178.

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

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

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

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

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

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

[10]. 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.

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

[12]. 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.

[13]. И. Бурдонов, А. Косачев. "Обход неизвестного графа коллективом автоматов". Труды Института системного программирования РАН, том 26-2, 2014 г., стр. 43-86.

[14]. И. Бурдонов, А. Косачев. "Исследование графа взаимодействующими автоматами". Новые информационные технологии в исследовании сложных структур. Материалы 10-ой российской конференции с международным участием. 2014, изд. Томского госуниверситета, стр.47-48.

[15]. И. Бурдонов, А. Косачев. "Параллельные вычисления на графе". Программирование, 2015, №1 (в печати).

# Graph Learning by a Set of Automata. The Nondeterministic Case

*Igor Burdonov <igor@ispras.ru>,*
*Alexander Kosachev <kos@ispras.ru>*
*Institute for System Programming of the Russian Academy of Sciences,*
*25, Alexander Solzhenitsyn st., Moscow, Russia, 109004.*

**Abstract.** Graph learning with automata is a basic task in many applications. Among such applications is formal model-based verification and testing of software and hardware systems, as well as network exploration including Internet and GRID. The model of a system or a network, in the final analysis, is reduced to an oriented graph of transitions with properties to be examined. In the recent years, the size of the real-life systems and networks and, consequently, the size of their graph models continuously grows. The problems arise when the graph learning by a single automaton (computer) either requires unacceptable long time, or the graph does not fit in the single computer memory, or both. Therefore, there is a task of parallel and distributed graph learning. This task is formalized as graph learning by a set of automata. The basis for such learning is graph traversal (traversing all its arcs accessible from the initial node). Automata can be generated in the initial node, move along the arcs of the graph according with their orientation and exchange messages through independent (of the graph) communication network. The total memory of the automata is used for storing the descriptions of the already traversed part of the graph. To move from node along an arc outgoing from it, the automaton should somehow identify this arc: indicate its number. In our paper "Graph learning by a set of automata" we proposed an algorithm of such traversal for deterministic graphs. The task is more complicated if the graph is nondeterministic. In such graph, to one arc number correspond, generally speaking, multiple arcs, one of which is chosen nondeterministically for traversal. To make the traversal possible, the following should be guaranteed: in unlimited number of experiments, each outgoing arc with the given number can be traversed. We call such nondeterminism fair. This paper covers the solution of the problem of traversal of fairly nondeterministic graphs.

**Keywords:** nondeterministic graphs, graph learning, graph traversal, communicating automata, parallel processing, distributed systems, testing.

**DOI:** 10.15514/ISPRAS-2015-27(1)-4

**For citation:** Burdonov Igor, Kosachev Alexander. Graph Learning by a Set of Automata. The Nondeterministic Case. *Trudy ISP RAN/Proc. ISP RAS*, vol. 27, issue 1, 2015, pp. 51-68 (in Russian). DOI: 10.15514/ISPRAS-2015-27(1)-4

## References:

[1]. R. Milner. A modal characterisation of observable machine-behaviour. In G. Astesiano & C. Bohm, editors: Proceedings CAAP 81, LNCS 112, Springer-Verlag, 1981, pp. 25-34.





[2]. van Glabbeek R.J. The linear time - branching time spectrum II; the semantics of sequential processes with silent moves. Proceedings CONCUR '93, Hildesheim, Germany, August 1993 (E. Best, ed.), LNCS 715, Springer-Verlag, 1993, pp. 66-81.

[3]. Y. Afek and E. Gafni, Distributed Algorithms for Unidirectional Networks, SIAM J. Comput., Vol. 23, No. 6, 1994, pp. 1152-1178.

[4]. 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.

[5]. 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.

[6]. 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.

[7]. 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.

[8]. 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.

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

[10]. 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).

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

[12]. 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).

[13]. Bourdonov I.B., Kossatchev A.S. Obhod neizvestnogo grafa kollektivom avtomatov [Unknown graph traversing by automata group]. Trudy ISP RAN [The proceeding of ISP RAS], Vol. 26-2, 2014, pp. 43-86 (in Russian).

[14]. Bourdonov I.B., Kossatchev A.S. Issledovanie grafa vzaimodejstvuyushhimi avtomatami. [ The graph learning by interacting automata] Novye informatsionnye tekhnologii v issledovanii slozhnykh struktur. Materialy 10-oj rossijskoj konferentsii s mezhdunarodnym uchastiem [New information technicks for complicated structure learning. The proceeding of 10-th Russian conference with international participation], Tomsk state university, 2014, pp. 47-48 (in Russian).

[15]. Bourdonov I.B., Kossatchev A.S. Parallel calculation on the graph. Programming and Computer Software, Vol. 41, No. 1, 2015 (in publication).