Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования

"Национальный исследовательский университет "Высшая школа экономики"

Факультет бизнес информатики отделение ПМиИ

Кафедра технологий моделирования сложных систем

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

На тему Формализация и анализ гибридных систем потока работ для решения научных задач.

Студент группы №271

Козлов Алексей Сергеевич

Научный руководитель

_____

(должность, звание, Ф.И.О.)

Москва, 2013 г.

1 Введение

За последних два десятилетия в научном сообществе компьютерное моделирование, названное e-Sience, стало незаменимой частью исследовательского процесса наравне с традиционными инструментами, такими как эмпирические, основанное на экспериментальных наблюдениях, теоретическое моделирование. Современная парадигма (e-Science) связана с развитием инструментария распределенных вычислений для научных исследований, позволяющего консолидировать вычислительные и программные ресурсы для решения сложных мультидисциплинарных задач в форме так называемых композитных приложений.

Термин e-Sience, возникший первоначально в Великобритании, где крупные исследовательские проекты в этой области начались в 2001г. Именно там было дано первое определение e-Science, получившее в дальнейшем широкое распространение: "научно-технологическая область, в которой всевозрастающую роль играет распределенное глобальное взаимодействие посредством сети интернет, с использованием очень больших коллекций данных, компьютерных ресурсов тера-уровня и высококачественной визуализации, доступных индивидуальному пользователю". В русском языке термин e-Science существует пока преимущественно в англоязычном варианте.
Необходимость в ,что кроме «обычной» информации, размещенной в интернете, специалисты, работающие в сфере науки и образования, нуждаются в доступе к крупномасштабным информационным массивам, базам данных, имеющим объемы памяти, измеряемые терабайтами. Работа с такими массивами требует вычислительных мощностей с производительностью уровня терафлоп. Задача e-Science, таким образом, — создание организационных и технологических структур, разработка соответствующего программного обеспечения для функционирования новой информационной среды с распределенными ресурсами (информационными и вычислительными), обеспечивающими доступ к ним индивидуальных пользователей, исследовательских групп, лабораторий и институтов. Основное русло реализации задач e-Science прокладывают грид-технологии. Эта концепция (нередко ее называют Grid Computing — распределенные сети вычислительных ресурсов) соответствует одному из ведущих и перспективных направлений развития информационно-коммуникационные

технологий. Наиболее распространенным подходом к представлению композитных приложений является формализм workflow.

Термин “workflow” используется в двух аспектах — как формальное представление некоторого процесса и как некоторый подход к автоматизации процессов, основанный на подобном представлении. Начнем с первого аспекта. Буквальный перевод термина “workflow” как «поток работ» плохо раскрывает содержание данного понятия, поэтому зачастую используются термины сценарий» или «рабочий процесс». Под workflow подразумевается формальное представление (модель) некоторого процесса, включающее в себя:

- Описание элементарных операций, из которых состоит процесс.
- Описание исполнителей, которые выполняют указанные операции.
- Описание зависимостей между операциями, а именно — потоков управления, которые определяют последовательность выполнения операций и синхронизацию между ними, и потоков данных, которые определяют передачу информации между операциями.
- Описание внешних событий, которые могут влиять на ход процесса, и правил их обработки.

Второй аспект термина “workflow” нашел отражение в определении, данном в Workflow Management Coalition [2], — « это автоматизация, полностью или частично, бизнес-процесса, при которой документы, информация или задания передаются для выполнения необходимых действий от одного участника к другому в соответствии с набором процедурных правил ». Данное определение, несмотря на привязку к бизнес-процессам, хорошо отражает суть workflow-методологии как некоторого подхода к автоматизации вообще говоря различных процессов. Системой управления workflow (workflow management system, WFMS) будем называть систему, позволяющую создавать сценарии, запускать и управлять их выполнением [2]. WFMS состоит из набора программных компонентов, предназначенных для хранения и интерпретации описаний процессов (сценариев), создания и управления экземплярами запущенных процессов, а также организации их взаимодействия с участниками процесса и внешними приложениями. Программное приложение, непосредственно выполняющее интерпретацию и запуск сценария, а также управляющее экземплярами за-

пущенных процессов, будем называть средой выполнения сценариев (workflow engine).

Приведем основные отличия научных вычислительных процессов от бизнес-процессов:

- Ориентация на сложные вычислительные ресурсы.
- Количество ресурсов, которые потребуются для выполнения сценария (решения задачи), может быть неизвестно априори, так как для некоторых классов задач трудно оценить предстоящий объем вычислений.
- Необходимость работы в динамичной распределенной среде, в которой ресурсы не известны априори и могут быть подвержены отказам.
- Работа с большими объемами данных.
- Необходимость выполнять большое количество идентичных заданий с переменными параметрами.
- Необходимость следить за выполнением процесса и контролировать его, в том числе внося специальные для конкретных случаев изменения.
- Для многих научных workflow характерны иерархии вложенных workflow, создаваемых и уничтожаемых по необходимости.

Указанные особенности научных вычислительных процессов определяют требования, которым должна удовлетворять WFMS, подходящая для описания и выполнения данных процессов в виде сценариев. Традиционные WFMS, рассчитанные на работу с бизнес-процессами, в подавляющем большинстве не подходят для решения научных задач. Поэтому, требуется разработка новых, научных WFMS, с одной стороны опирающихся на сформировавшуюся workflow-методологию, а с другой стороны — специально рассчитанных на требования научных приложений. Пожалуй, главное, что могут почерпнуть научные WFMS из накопленного в данной области опыта, это способы формального представления workflow. В следующем разделе мы рассмотрим основные подходы к представлению сценариев, ссылаясь на уже созданных научных WFMS для того, чтобы одновременно дать обзор существующих решений в области научных вычислительных workflow.

Мотивация

Целью работы является построение модели workflow, ориентированного на решение научно-инженерных задач в распределенной вычислительной среде. Модель разрабатывается с целью получения более естественного способа описания реальных инженерных и научных задач. К целям работы так же относится разработка методов анализа построенной модели. Средства анализа разрабатываемые в работы:

- проверка построенных сценариев на ошибки.
- методология выработки наилучшей стратегии запуска сценария в распределённой вычислении среде на основе эвристик.

Программная реализация модели и средств её анализа будет представлена в виде модуля на языке программирования Python.

2 Обзор существующих моделей workflow

2.1 Модели управления запуском workflow

Большинство моделей управления workflow можно разделить на два класса: модели ориентированные на потоки данных(data-flows) и модели ориентированные на потоки управления(control-flows). Оба класса определяют взаимодействие между отдельными задачами-компонентами workflow, но различаются принципами реализации этого взаимодействия.
В workflow, основанных на принципе control-flow, связи между элементами workflow представляет передачу управления от одного задания следующему. Это позволяет формировать внутри workflow такие типы структур, как: последовательное выполнение, параллельное выполнение, циклическое выполнение и условные переходы. в workflow, основанные на принципе data-flow, зависимости между элементами workflow, определяют направление потоков данных от

Так же существуют гибридные системы управления workflow, сочетающие в себе принципы обоих приведённых выше классов. Гибридные системы поддерживают оба типа зависимостей между компонентами workflow, но один из типов, как правило, является доминирующим, а другой используется при необходимости в особых

случаях. Например, в data-flows системе такой как Triana, возникают ситуации, когда необходимо последовательно связать задачу, не производящую никакие данные, с задачей, не требующей данных на вход. В таком случае, на этом участке будет использован переход к control-flow зависимости.

Существует много техник построения расписаний запуска workflow. Но почти все они применимы, только если worklow представим в виде направленного ациклического графа (DAG) заданий. В случае, если workflow содержит циклы, параллельные циклы или условные переходы, применяются методы приведения графа заданий workflow к требуемому виду. С подобной проблемой преобразования графа задач уже сталкивались при разработке компиляторов, поддерживающих автоматическую параллелизацию, и были выработаны техники развёртывания параллельных циклов, устранения обычных циклов, предсказаний при условных переходов.

Примеры приложений, использующих DAG для представления workflow: Condor [10], Symphony [11], Cactus [12], UNICORE [13].

2.2 Различные подходы к представлению workflow

За время существования методологии workflow возникло несколько различных подходов их формального описания:

- Использование скриптовых языков;
- Использование графов. Наиболее часто используемые типы графов:
  (a) ориентированные ациклические графы(DAG);
  (b) сети Петри.

2.2.1 Скриптовые языки

Скрипт(script) представляет собой набор команд, предназначенный для выполнения интерпретатором без вмешательства пользователя. Скриптовые языки в качестве средства представления сценариев могут быть удобны пользователям, имеющим опыт программирования. Но не смотря на высокий уровень и простоту, они не достаточно наглядны и интуитивны для обычных пользователей.

В качестве примера успешного использования скриптового языка для конструирования сценариев стоит упомянуть систему GridAnt [5], Karajan [6].

2.2.2 Представление сценариев в виде графов

Другой способ представления workflow - это изображение в виде графа. Изначально, графы - это чисто математическая абстракция, но тем не менее, они удобны для неподготовленного пользователя, поскольку представляют workflow наглядно. Правда с увеличением сложности workflow графы тоже усложняются и их становится тяжелее просматривать, тем не менее, наглядность можно сохранить, используя иерархическое представление графа, позволяющее скрывать детали отдельных подграфов.
Для представления workflow наиболее часто используются два типа графов: ориентированные ациклические графы(DAG) и сети Петри.

2.2.3 Ориентированный ациклический граф

Ориентированным ациклическим графом называется любой ориентированный граф, в котором нет ориентированных циклов [8]. Вершинами графа являются, например, исполняемые программы или выполняемые операции, а ребра устанавливают зависимости между ними. Преимущество таких графов — простота структуры и реализации. Но есть и недостатки: они накладывают ограничения на типы сценариев — например, нельзя явно задать циклы без применения дополнительных конструкций, уже не связанных с графовым представлением. Кроме того, такие графы способны описывать только модель поведения процесса, не фиксируя его состояние во время выполнения.

2.2.4 Сети Петри

Сети Петри — особый класс ориентированных графов. Теория сетей Петри является хорошо известным и популярным формализмом, предназначенным для работы с параллельными и асинхронными системами. Основанная в начале 60-х гг. немецким математиком К. А. Петри, в настоящее время она содержит большое количество моделей, методов и средств анализа.

Рассмотрим основные преимущества сетей Петри при представлении workflow.

Графическая природа. Сети Петри — графический язык. Поэтому они интуитивно понятны и легки для изучения. Их графическая природа также удобна для взаимодействия с конечными пользователями.

Формальность описания. Сценарий, описанный в терминах сети Петри, задается строго и точно, потому что семантика классических сетей Петри, как и некоторых дополнений к ним (цвет, время, иерархия) были введены формально.

Выразительность. Сети Петри поддерживают все базисные элементы, необходимые для описания сценария. С их помощью могут быть смоделированы все управляющие конструкции, существующие в современных системах управления сценариями. Более того, точное представление состояний сценария позволяет описывать ситуации неявного выбора и сохранять промежуточные состояния, с возможностью возвращения к ним.

Свойства. В последние десятилетия были подробно изучены основные свойства сетей Петри. Прочные математические основания позволяют делать строгие выводы из этих свойств.

Анализ. Сети Петри отличаются наличием большого числа методов анализа. Это их ценное преимущество с точки зрения использования для описания сценариев — данные методы могут быть использованы для доказательства различных свойств (выполнимости, инвариантности, мертвых переходов и т. д.) и для вычисления характеристик выполнения сценариев (время отклика, время ожидания, степень занятости и пр.). Таким образом, становится возможным оценивать альтернативные сценарии, используя традиционные инструменты анализа сетей Петри.

Подробнее сети Петри, их свойства и средства анализа будут описаны в следующей главе.

2.2.5 Модели, ориентированные на потоки данных

Помимо уже упомянутых методов, существует еще один подход, в котором для представления сценариев могут использоваться как скриптовые языки, так и графы, но который несколько отличается от обсуждавшихся ранее подходов своей спецификой. Речь идет о системах управления научными вычислительными сценариями, ориен-

тированными на потоки данных. Как уже было сказано, в научных приложениях часто все необходимые действия сводятся к различным операциям над данными, т. е. в подобных процессах потоки управления и потоки данных совпадают. Поэтому нет необходимости вводить специальные элементы языка для описания логических конструкций, а достаточно просто обеспечить средства объединения элементарных модулей обработки данных в сеть. Каждый модуль имеет один или несколько входов и выходов, дуги сети соответствуют соединениям выхода одного модуля с входом другого, по которым осуществляется передача данных между модулями. Как только на вход модуля поступили все необходимые данные, происходит запуск программного кода модуля, который производит обработку входных данных, после чего полученные результаты (выходные данные) помещаются в выходы модуля и передаются по соединениям на вход других модулей.

Пример систем, обеспечивающих описанную функциональность: Triana [14], Kepler [15], Taverna [16].

Этот подход к представлению workflow и будет использоваться в данной работе.

3 Формальное описание разрабатываемой модели workflow

Как уже было сказано выше, за основу разрабатываемой модели будет взято возможность представления workflow в виде сети, состоящей из модулей, имеющих интерфейс в виде входов и выходов(именуемых в нашей работе портами), и каналов связи(именуемых в нашей работе связями), соединяющих входы и выходы модулей. После по того как данные поступили на все необходимые для начала работы модуля входы, модуль начинает работу, по окончанию работы обработанные данные помещаются в выходные порты модуля.

Новизна предлагаемого подхода заключается в том, что каждый модуль может иметь несколько состояний,каждое из которых определяет требования к набору необходимых для запуска входов. После завершения работы модуль так же может отправить обработанные данные по одному из характерных для этого состояния набору выходов и изменить своё состояние.

3.1 Представление workflow

В нашей модели схема workflow задаётся графом связей. Граф связей workflow $WFG = (A, D)$ состоит из конечного и непустого набора вершин A, представляющих блоки и набора рёбер D , представляющих связи, соединяющие блоки через порты.
При этом в состав набора блоков всегда A входят два уникальных абстрактных блока *Source* и *Stock*, вводимых для задания точек запуска и завершения workflow.

3.1.1 Связи

В нашей модели связи - абстракция передачи данных, считаем, так же будем считать, что данные между блоками передаются мгновенно и без потерь.

Формально связь $d \in D$ представима в виде четвёрки (s, p_s, t, p_t), где $s, t \in A$ - идентификаторы входной и выходных блоков, $p_s \in out(s), p_t \in in(p)$ - выходной и входной порты соответствующих блоков.

Более того в рассматриваемой модели мы абстрагируемся от типизации данных передаваемых между блоками, считая что для любых двух соединённых блоков типы данных для каждой пары соединённых портов совместимы. Поэтому мы сразу можем представлять передаваемые по связям данные как сигналы.

Опишем некоторые свойства связей:

- Связь может быть установлена только между выходным портом одного блока и входным портом другого блока или самого себя.
- Связи могут быть построены как из одного выходного порта ко многим входным, так и из нескольких выходных в один входной порт.
- Если блок испускает сигнал по какому либо порту, то сигнал распространяется по всем связям исходящих из этого порта, что позволяет моделировать параллельный запуск нескольких блоков.
- Входные и выходные порты блоков могут оставаться неподключенными.

3.1.2 Блок

Блок в нашей модели является единицей исполнения workflow . В рассматриваемой модели считаем, что число состояний блока конечно, и набор входных и выходных портов фиксирован. Тогда поведение блока можно формализовать в виде недетерминированного конечного автомата.

Автоматом называется система, выходы которой зависят не только от поступивших входов, то и от текущего состояния системы. Состояние системы может быть обозначено переменной состояния $s \in \Sigma$, где Σ - это набор всех возможных состояний системы. Конечным автоматом называется автомат, для которого число состояний Σ конечно.

Классическая теория конечных автоматов (Hopcroft and Ullman, 1979) различает два вида автоматов: **Автомат Мили** и **Автомат Мура**. В Автомате Мили выходное значения сигнала явно зависит только от входных значений, в отличии от Автомата Мура, выходное значение сигнала в котором зависит лишь от текущего состояния данного автомата. Для полного задания автомата Мили или Мура дополнительно к законам функционирования, необходимо указать начальное состояние и определить внутренний, входной и выходной алфавиты. Между автоматами Мили и Мура существует соответствие, позволяющее преобразовать закон функционирования одного из них в другой или обратно. Автомат Мура можно рассматривать как частный случай автомата Мили, имея в виду, что последовательность состояний выходов автомата Мили опережает на один такт последовательность состояний выходов автомата Мура, т.е различие между автоматами Мили и Мура состоит в том, что в автоматах Мили состояние выхода возникает одновременно с вызывающим его состоянием входа, а в автоматах Мура - с задержкой на один такт, т.к в автоматах Мура входные сигналы изменяют только состояние автомата.

Определим недетерминированный автомат атомарного блока, который будет использоваться в работе.

Определение: Конечный автомат блока *Конечным автоматом блока называется набор* $M = (\Sigma, I, O, T, s_0)$ *, где*

- Σ *-набор конечных состояния,*

- I *- непустой набор доступных входных портов блока,*

- O *- непустой набор доступных выходных портов блока,*

- $I \bigcap O = \varnothing$ -

- $s_0 \in \Sigma$ *- начальное состояние,*

- $T : \Sigma \times (2^I \backslash \{\varnothing\}) \to 2^{\Sigma \times 2^O}$ *отображение сопоставляющее каждому состоянию и набору входных портов набор состояний с соответствующим набором выходных портов.*

Если для заданного состояния $s' \in \Sigma$ и непустого набора входных портов $in \in 2^I \backslash \{\varnothing\}$ отображение T(s', in) не пусто, то результат отображения T(s', in) является набором возможных вариантов работы блока, с поглощением сигналов(данных) на входных портах, соответствующим переходом в новое состояние и отправлением сигналов по указанным выходным портам. Если же T(s', in) даёт пусто множество, считается, что условия запуска блока из состояния s' не соблюдены и начало работы программного модуля невозможно.

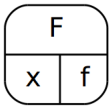
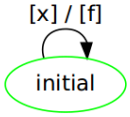
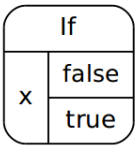
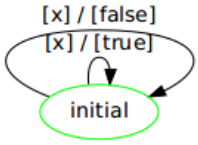
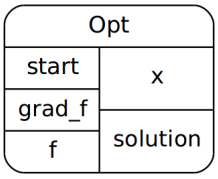
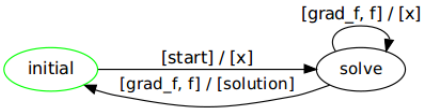
Как уже было определено выше, у блока имеется **начальное состояние** s_0, т.е. то в котором автомат блока находится перед запуском.

Визуально состояния соединены переходами, рядом с которыми указано, какие порты необходимы для срабатывания перехода и сигналы по каким портам будут испущены.

В Таблице 1. приведены примеры блоков.

c|c|>m5cm l

Таблица 1: Примеры блоков. Для каждого блока представлен его конечный автомат в виде ориентированного графа. Над каждым ребром графа, обозначающим переход из одного состояния в другое или в само себя, указан набор входных портов необходимый для осуществления перехода и набор выходных портов, по которому будут отправлены сигналы.

Тип блока	Конечный автомат блока	Описание
		Простой блок - функция. Имеет всего одно состояние, принимаем сигнал по порту x и отправляет сигнал по порту f.
		Условный блок с единственным состоянием, требующий только сигнал на порту x, но по окончанию работы отправляет сигнал либо через порт true, либо через порт false.
		Оптимизирующий блок с начальным состоянием initial и переходящий в состояния solve после получения сигнала на порт start. Из состояния solve, принимая сигналы с двух портов, может как остаться в этом же состоянии, так и вернутся в исходное состояние, отправив сигнал по порту solution.

4 Возможные ошибки при построении workflow

Так как в нашей модели мы подразумеваем возможность асинхронной работы блоков , то как и в многопоточных компьютерных программах возможно возникновение состояния неопределённости, известной как *состояние гонки*(Race condition). В программировании состояние гонки определяется как ситуация, когда несколько потоков одновременно обращаются к одному и тому же ресурсу, причём хотя бы один из потоков выполняет операцию записи, и порядок этих обращений точно не определен.

Неопределенности подобного рода мы будем рассматривать как ошибки, обнаружению которых посвящена часть данной работы. Такого рода неоднозначности, особенно в реальных системах, то есть в условиях неопределенного времени вычислений блоков и их параллельного выполнения, может привести к совершенно иному ходу потока данных, нежели было задумано автором потока данных. Поэтому наличие подобного рода неопределенностей (состояний гонки) говорит скорее о неправильном построении схемы workflow, чем о некой задумке автора.

Рассмотрим некоторые возможные случаи неопределённого поведения контексте нашей модели workflow, которые можно обнаружить ещё на этапе построении сценария workflow:

- На один входной порт могут прийти два сигнала одновременно.
- На входные порты блока одновременно приходит такой набор сигналов, что запуск блока возможен по разным наборам сигналов. Такое состояние гонки потенциально возможно для блоков, для которых выполняется:
  $(S, \Sigma, \Lambda, T, s_0)$ - автомат блока;
  $\exists s \in S, \exists in1, in2 \in 2^{\Sigma} \backslash \{\varnothing\}, in1 \neq in2 :$
  $T(s, in1) \neq \varnothing \bigwedge T(s, in2) \neq \varnothing.$

Более подробно и строго об неоднозначностях в поведении потока данных будет сказано при описании алгоритма, моделирующего работу workflow.

4.1 Принципы моделирования работы workflow

Для построенной модели workflow, был построен алгоритм, имитирующий работу построенного сценария workflow и позволяющий обнаружить ошибки, допущенные при построении, такие как состояния гонки.

Прежде чем начать описание самого алгоритма, определим условия, характеризующие начало и завершение работы сценария workflow.

- workflow начинает работу, испуская сигналы из всех выходных портов блока Source;
- workflow заканчивает работу, как только на все входные порты блока Stock поступили сигналы, после чего они поглощаются.

Мы считаем, что workflow построен корректно если выполняются все три правила:

- в ходе выполнения этой workflow невозможен случай возникновения состояние гонки;
- workflow всегда завершается;
- при завершении работы workflow, не остаётся непоглощённых сигналов.

Такие workflow мы будем назвать корректными.

Для корректных workflow (при требовании отсутствия состояний гонки) порядок запуска блоков не зависит от времени их работы, а только от распространения сигналов, будем считать, что любой блок в любом состоянии выполняется фиксированное время $T > 0$.

Для описания алгоритма работы workflow потребуется ввести некоторые дополнительные понятия, позволяющие определить его динамические характеристики.
Понятие разбиения потока вводится для формализации понятия конкуренции между сигналами, распространяющимися по связям внутри workflow.

Опредление: Разбиение потока *Разбиением потока назовём пару* ***(fork, split)****, где fork - имя блока, которой испустившего*

набор сигналов, а split - кортеж состоящий из n элементов, где n - число исходящих от блока сигналов. А каждый элемент кортежа может быть либо 1, либо 0, либо другим разбиением.

Опредление: Нулевое разбиение *Под нулевым разбиением понимаем разбиение с пустым параметром fork и кортежем splitting из одного элемента т.е. (, (1))*

Далее введём некоторый набор операций, доступный над разбиениями, задаваемых в виде рекурсивных функций. В работе функции приведены в виде таблиц, где название столбца - значение первого аргумента функции, название строки - значение второго аргумента функции, а их пересечение - возвращаемое значение.

- **Произведение (×)** двух разбиений задаётся рекурсивной функцией mult от двух аргументов, возвращающей новое разбиение. Табл.2

Таблица 2: Функции умножения двух разбиений, представленная в виде таблицы.

	WS1	1	0
WS2	(WS1.fork, ([(WS1.split[i] × WS2) for i in [0,...,n]]))	WS2	0

- **Сложение (+)** двух разбиений задаётся рекурсивной функцией sum от двух аргументов, возвращающей новое разбиение. Табл.3

Таблица 3: Функции сложения двух разбиений, представленная в виде таблицы. throw() обозначает недопустимость операции при данных аргументах.

	WS1	1	0
WS2	if WS1.fork == WS2.fork then (WS1.fork, ([(WS1.split[i] + WS2.split[i]) for i in [0,...,n]])) else throw()	throw()	WS2
1	throw()	throw()	1
0	WS1	1	throw()

- **Проверка на конкуренцию ($\perp$)** задаётся рекурсивной функцией concurrent от двух аргументов, возвращающей True в случае выявления конкуренции и False в противном случае. Табл.4

Таблица 4: Функция проверки двух разбиений на конкуренцию, представленная в виде таблицы.

	WS1	1	0
WS2	if WS1.fork == WS2.fork then all([(WS1.split[i] $\perp$ WS2.split[i]) for i in [0,...,n]]) else False	False	True
1	False	False	True
0	True	True	False

В каких случаях используются разбиения и применяются данные операции над ними будет определено в дальнейшем. Для наглядности приведём несколько примеров применения этих операций над разбиениями:

Умножение двух разбиений:

$$(,(1)) \times (b1,(0,1)) = (,(b1,(1,0)));$$

$$(,(b1,(1,0,1))) \times (b2,(0,1)) = (,(b1,((b2,(0,1)),0,(b2,(0,1)))));$$

Сложение двух разбиений:

$$(,(b1,(1,0))) + (,(b1,(0,1))) = (,(b1,(1,1)));$$

При сложении двух разбиений так же применяется рекурсивное **правило упрощения**, которое определяется как:
Пусть дано разбиение WS = (fork, split), у которого кортеж split состоит из одних единиц и WS не является нулевым разбиением, тогда $WS \equiv 1$:

$$(,(b1,(1,0))) + (,(b1,(0,1))) = (,(b1,(1,1))) \equiv (,(1));$$

Проверка двух разбиений на конкуренцию:

$$(,(b1,(1,0))) \perp (,(b1,(0,1))) = True$$

$$(,(1)) \perp (,(b1,(0,1))) = False$$

Так же введём понятие волны. Волна представляет представляет распространение сигнала по связи, т.е. испущенного одним блоком, но ещё не поглощённого другим, и некоторую информацию в виде разбиения, позволяющую в определить, может ли этот сигнал конкурировать с другими.

Определение: Волна(Wave) *Волной внутри workflow, заданного графом связей WFG = (A,D), будем называть пару вида (connection, wavesplit) , где connection $\in D$ - ребро графа связей, wavesplit - некоторое разбиение.*

Определение: состояние workflow (Workflow State) *Для worklow с заданным графом связей WFG = (A,D), состояние определяется как набор (BlockStates, WaveFront, BlockHistory)*

- *BlocksStates - отображение $BS : a \to s_a, a \in A, s_a \in states(a)$, сопоставляющая каждому блоку в составе workflow его текущее состояние.*

- *WaveFront - набор волн.*

- *BlockHistory - отображение $H : a \to splits, a \in A$, сопоставляющая каждому блоку набор значений разбиений потока splits, прошедших через этот блок.*

При этом идентификатором уникальности любого состояния workflow WFS = (bs, front, hist) будет только двойка (bs, edges), где edges - набор рёбер $\{d : (d, ws) \in front\}$.

Определение: начальное состояние workflow (Initial Workflow State) *Начальным состоянием workflow будем называть то состояние textit($bs_{init}, front_{init}, hist_{init}$), в котором каждый блок находится в начальном состоянии, в соответствии с его конечным автоматом, и набор волн $front_{init}$, соответствующий испущенным сигналам из выходных портов блока Source. А отображение $hist_{init}$ определяется как: $hist_{init} : a \to \varnothing, \forall a \in A$.*

4.2 Алгоритм проверки workflow описанного в рамках введённой модели

Суть разработанного алгоритма заключается в нахождении всех состояний workflow, достижимых из начального состояния workflow и проверке на возникновение состояния гонки на каждом шаге алгоритма.
Алгоритм считается дискретным, с шагом по времени равным T, т.е. времени работы любого из блока.
Таким образом на каждом шаге работы алгоритма,для всех уникальных состояний workflow, полученных на предыдущем шаге , применяется функция GenerateStates. Работа алгоритма начинается с применения функции GenerateStates к начальному состоянию workflow на первом шаге. Завершение работы алгоритма происходит, когда достигается одно из условий:

- алгоритма прерывается сообщением об обнаружении состояния гонки.

- за новый шаг алгоритма не получено ни одного нового уникального состояния.

Функция GenerateStates ***нахождения достижимых за 1 шаг состояний workflow из состояния передаваемого на вход.***

Входной параметр:

CurrentState = (bs, front, hist) - состояние workflow .

Алгоритм:

1. *По информации о всех распространяющимся сигналам, полученной из front, получаем на каких портах каждого из блоков сигналы ждут обработки.*

2. *Далее для каждого блока, на основе значения его состояния из bs, b набора входных портов с сигналами, полученным на шаге 1. и функции отображения его автомата получаем набор возможных вариантов его работы. Где каждый вариант работы это связка вида (in, out, nextState), т.е. набор входных портов*

in, сигналы которых он поглотит, состояние nextState в которое он перейдёт и набор выходных портов out, сигналы по которым он отправит. Если для блока множество вариантов работы пусто, это означает, что запустится он сейчас не может.

3. *Находим прямое произведение наборов вариантов работы всех блоков, полученных на шаге 2 и имеющих непустое множество вариантов работы. Т.е. множество вариантов одновременной работы всех блоков, готовых к запуску.*

4. *Для каждого варианта одновременной работы блоков из полученных прямым произведением на шаге 3, рекурсивно применим функцию EvolveState.*

Возвращаемое значение: *Набор состояний состояний достижимых из состояния данного на вход функции.*

Функция EvolveState ***преобразующая задаваемое на вход состояние workflow при известном результате работы блока.***
Входные параметры:

TargetState = (bs, front, hist) - состояние workflow

WorkVariant = (a, in, out, state) - вариант запуска блока, где a - идентификатор блока, in - набор входных портов блока, сигналы по которым он поглотит, в результате работы , out - набор выходных портов блока сигналы по которым будут испущены, state - состояние в которое он перейдёт.

Алгоритм:

Находим $waves_{in}$ *- набор волн из front, связи которых соединение с портами in блока a.*
$edges_{out}$ *- набор связей, выходящих из портов out,* $n_{out} = |edges_{out}|$ *- число связей.*

Параметр разбиения каждой волны из $waves_{in}$ *проверяем на конкурентность с разбиениями прошедшими, через этот блок. В случае, если какая либо пара разбиений окажется конкурентной, означает, что обнаружено потенциальное состояние гонки , что мы считаем ошибкой в композиции workflow. Это событие вызывает прерывание работы всего алгоритма и выводом сообщения об*

ошибке с детализированной информацией о месте возникновения состояния гонки. Формальная запись условия отсутствия состояний гонки:

$$\forall_{w \in waves_{in}} \forall_{split_{prev} \in hist(a,p_{in})} w.split \perp split_{prev} = False \qquad (1)$$

Если ошибок не обнаружено, то рассчитаем промежуточное значение разбиения потока:
$ws_{sum} = \sum_{w \in waves_{in}} w.split)$

Теперь считаем значения выходных волн:
$waves_{out} = \{(edges_{out}[i], ws_{sum} \times (a, 1_{i,n_{out}}) : i \in [1, n_{out}]\}$.

Под $1_{i,n}$ *понимаем кортеж, состоящий из n элементов, все элементы которого кроме i-го равны 0, а i-ий элемент 1.*

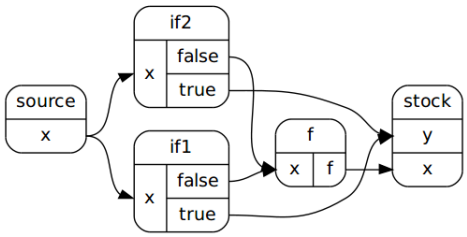
bs_{new} *= bs с изменённым значением bs(a) = state.*
$front_{new} = (front\ waves_{in}) \cup waves_{out}$. $hist_{new}$ *= hist с дополненной информацией пройденных через порты разбиениями потока.*

Возвращаемое значение: *изменённое состояние workflow (*bs_{new}*, front, hist)*

Приведённый алгоритм конечен, т.к. останавливается при нахождении всех достижимых уникальных состояний workflow. Для workflow с графом связей (A, D) число состояний не превышает $|D|! \prod_{a \in A} c_a$,где c_a - конечное число состояний блока a.

На основе получаемых в ходе работы алгоритма состояний workflow может быть построен ориентированный граф состояний workflow, где набор вершин - все найденные уникальные состояния, а ориентированные рёбра - переходы из одного состояния в другое за 1 шаг.

Рис. 1: Пример некорректно составленного workflow. Блоки If1 и If2 начинают работу асинхронно, при этом в случае, когда оба блока типа If в результате своей работы отправляют сигналы



Как уже было сказано, что для корректных workflow можно построить ориентированный граф состояний, представляющий схему работы workflow. С помощью несложных преобразований можно привести полученный граф состояний к виду сети Петри, тем самым мы получим более наглядное представление о динамике системы и возможность использовать уже разработанные для сетей Петри техники анализа. В следующей главе будут рассмотрены сети Петри, их свойства и некоторые инструменты анализа.

5 Анализ сетей Петри

Сети Петри широко используются для моделирования и исследования динамических дискретных систем. Сеть Петри представляет собой двудольный ориентированный граф, состоящий из вершин двух типов — *позиций* и *переходов*, соединённых между собой дугами. Вершины одного типа не могут быть соединены непосредственно. В позициях могут размещаться метки (маркеры), способные перемещаться по сети.

Определение: Сеть Петри *Простой сетью Петри называется набор* $PN = (S, T, F)$ *, где*

1. $S = \{s_1, \ldots, s_n\}$ *- множество позиций*
2. $T = \{t_1, \ldots, t_r\}$ *- множество переходов таких, что* $S \bigcap T = \varnothing$.
3. $F \subseteq \mu S \times T \times \mu S$ *- отношение инцидентности такое, что*
   - $\forall \langle Q_1^{'}, t_1, Q_1^{''} \rangle, \langle Q_2^{'}, t_1, Q_2^{''} \rangle \in F : \langle Q_1^{'}, t_1, Q_1^{''} \rangle \neq \langle Q_2^{''}, t_2, Q_2^{'} \rangle \Rightarrow t_1 \neq t_2$;
   - $\{t | \langle Q_1^{'}, t_1, Q_1^{''} \rangle \in F\} = T$

Условия в пункте 3 говорят , что для каждого перехода $t \in T$ существует единственный элемент $\langle Q^{'}, t, Q^{''} \rangle$, задающий для него входное множество $Q^{'}$ и выходное множество $Q^{''}$. Дадим определение входному и выходному множеству.

Определение: Входное и выходное множества позиций и переходов

Пусть задана сеть $N = (S, T, F)$.

1. *Если для некоторого перехода t имеем* $\langle Q^{'}, t, Q^{''}\rangle \in F$ *, то будем обозначать*
   $\bullet t = Q^{'} = \langle s | t \in s\bullet\rangle, t\bullet = Q^{''} = \langle s | t \in \bullet s\rangle$

2. *И соответственно*
   $\bullet s = \langle t | s \in t\bullet\rangle, s\bullet = \langle t | s \in \bullet t\rangle$

Будем говорить, что $\bullet t$ *- входные , а* $t\bullet$ *- выходные позиции для перехода t. Таким образом, согласно определению, справедливо:* $\forall t \in T : \langle \bullet t, t, t\bullet\rangle \in F$.
Позиция s называться инцидентной переходу t , если $s \in \bullet t$ *или* $s \in t\bullet$.

Сети Петри имеют удобную графическую форму представления в виде графа, в котором позиции изображаются кружками, а переходы прямоугольниками. Причем позиция s соединяется с переходом t если $s \in \bullet t$ и t соединяется с s если $s \in t\bullet$.

Само по себе понятие сети имеет статическую природу. Для задания динамических характеристик используется понятие маркировки сети $M \in \mu S$, т.е. функции $M : S \longrightarrow N_0$, сопоставляющей каждой позиции целое число. Графически маркировка изображается в виде точек, называемых метками (tokens), и располагающихся в кружках, соответствующих позициям сети. Отсутствие меток в некоторой позиции говорит о нулевой маркировке этой позиции.

Определение: Маркированная сеть Петри *Маркированной сетью Петри называется набор* (PN, M_0)*, где*

1. *PN =(S,T,F) - сеть;*

2. $M_0 \in \mu S$ *- начальная маркировка.*

Сети Петри были разработаны и используются для моделирования параллельных и асинхронных систем. При моделировании в сетях Петри позиции символизируют какое-либо состояние системы, а переход символизируют какие-то действия, происходящие в системе. Система, находясь в каком-то состоянии, может порождать определенные действия, и наоборот, выполнение какого-то действия переводит систему из одного состояния в другое.

Текущее состояние системы определяет маркировка сети Петри, т.е. расположение меток (токенов) в позициях сети, т.е. $M \in S \rightarrow N$. Под маркировкой, представленной в виде: $1s_1 + 2s_2 + 1s_3 + 0s_4$,

понимаем, что в позиции s_1 находится 1 метка, 2 метки в s_2, 1 метка в s_3 и ни одной метки в s_4. Так же представление этой маркировки можно сократить до $1s_1+2s_2+1s_3$. Для сравнения двух маркировок, мы определяем частичный порядок.

$$\forall_{M_1,M_2} M_1 \leq M_2, \forall_{s\in S} : M_1(s) \leq M_2(s).$$

Выполнение действия в системе, в сетях Петри определяется как срабатывание переходов. Срабатывание переходов порождает новую маркировку, т.е. порождает новое размещение меток (токенов) в сети.

Работа мариковочной сети Петри управляется наличием или отсутствием маркировочных меток. В сети Петри срабатывает переход t, в процессе которого с каждого входа $s \in \bullet t$ снимается метка и каждому выходу $s \in t\bullet$ прибавляется метка.

Определение: Правило срабатывания переходов *Пусть* $\Sigma = (S, T, F, M_0)$ *маркировочная сеть.*

1. *Переход* $t \in T$ *считается возбуждённым при маркировке* $M \in \mu S$*, если каждое положение* $s \in \bullet t$ *имеет хотя бы одна метка;*

2. *Переход t, возбуждённый при маркировке M, может сработать, приведя к новой маркировке* $M^{'}$*, которая получается путём поглощения метки на каждом позиции* $s \in \bullet t$ *и появления новой метки на каждой позиции* $s' \in t\bullet$*.*

Для сети Петри (S, T, F) и её текущей маркировки $M_1 \in \mu S$ опишем следующие обозначения:

- $M_1 \xrightarrow{t} M_2$: преход t при маркировке M_1 возбуждён и при его срабатывание приводит к маркировке M_2

- $M_1 \rightarrow M_2$: существует переход t, такой что $M_1 \xrightarrow{t} M_2$

- $M_1 \xrightarrow{\sigma} M_n$: последовательность переходов $\sigma = t_1t_2t_3...t_{n-1} \in T^*$ переводящая маркировку M_1 в M_n через набор (возможно пустой) промежуточных маркировок $M_2, ..., M_{n-1}$, т.е.: $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_1} ... \xrightarrow{t_{n-1}} M_n$
  Маркировка M_n называется *достижимой* из M_1 (и имеет обозначение $M_1 \xrightarrow{*} M_n$) тогда и только тогда, когда существует

последовательность такая переходов σ, что $M_1 \xrightarrow{\sigma} M_n$. Отметим, что пустая последовательность переходов тоже допустима, $M_1 \xrightarrow{*} M_1$.
Когда же мы имеем сеть Петри с начальной маркировкой (S, T, F, M_0), то маркировка $M^{'}$ будет допустимой, если $M_0 \xrightarrow{*} M^{'}$.

Определение: Живость(Live) *Маркированная сеть Петри (PN, M) называется живой, если для любой достижимой маркировки $M^{'}$ и любого перехода t существует маркировка $M^{''}$ достижимая из маркировки $M^{'}$ и задействующая переход t.*

Определение: Ограниченность(Bounded), безопасность(Safe) *Маркированная сеть Петри (PN, M) называется ограниченной, если для каждого положения s существует такое натуральное число n, что для каждой достижимой маркировки меток в этом положении меньше n. Сеть называется безопасной, если максимальное число меток не превышает 1. Безопасность является предпочтительным свойством, т.к. нелогично было бы допускать несколько меток в позиции, представляющей состояние.*

Определение: Правильность(Well-formed) *Сеть Петри PN называется правильной, если существует такая начальная маркировка M_0, что сеть(PN, M_0) будет живой и ограниченной.*

Определение: Путь *Пусть дана сеть Петри PN. Путь C из вершины n_1 в вершину n_k представляет собой последовательность вершин $\langle n_1, n_2, ..., n_k \rangle$ такую, что $\langle n_i, n_{i+1}, n_{i+2} \rangle \in F$ для $1 \leq i \leq k-2$.*
Путь C называется простым, если для любых двух вершин $n_i, n_j \in C$ выполняется: $i \neq j \Rightarrow n_i \neq n_j$.
Путь C называется бесконфликтным, если для любой положения n_j и любого перехода n_i , $n_i, n_j \in C$, выполняется $j \neq i-1 \Rightarrow n_j \notin \bullet n_i$.
Для удобства введём оператор α над путями. Тогда для пути $C = \langle n_1, n_2, ..., n_k \rangle$ $\alpha(C) = \{n_1, n_2, ..., n_k\}$

Определение: Сильная связность(Strongly Connected) *Сеть Петри называется сильно связной, если для любой пары вершин x и y существует путь из x в y.*

Определение: Свободный выбор(Free-Choice) *Сеть Петри обладает свойством свободного выбора, если для любых двух переходов $t_1, t_2 \in T$ выполняется $\bullet t_1 \cap \bullet t_2 \neq \varnothing \Rightarrow \bullet t_1 = \bullet t_2$*

Определение: WF-сеть(Workflow-net) *Пусть дана сеть*

Петри PN = (S,T,F).

- *Если PN - WF-сеть с входной позицией i, тогда для любой позиции $s \in S : \bullet s \neq \{s = i$, т.е. i -единственная входная позиция.*

- *Если PN - WF-сеть с выходной позицией o, тогда для любой позиции $s \in S : s\bullet \neq \{s = o$, т.е. o -единственная выходная позиция.*

- *Если PN - WF-сеть и мы добавляем в PN переход t^*, соединяющий выходную позицию o и входную позицию i (т.е. $\bullet t^* = o, t^*\bullet = i$), тогда получившаяся сеть будет обладать сильной связностью.*

Для WF-сети начальную маркировку с единственной меткой только во входной позиции i будем обозначать $_i$, а маркировку с единственной меткой только в выходной позиции o соответственно $_o$.

Определение: Корректность(Soundness) *WF-сеть PN = (S,T,F) корректна тогда и только тогда, когда:*

(i) Для каждой маркировки M ,достижимой из i, маркировка o так же достижима из M.

$$\forall M(i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} o)$$

(ii) Маркировка o - единственная достижимая из i маркировка, имеющая хотя бы одну метку в позиции o.

$$\forall_M(i \xrightarrow{*} M \wedge M \geq o) \Rightarrow (M = o)$$

(iii) В (PN,i) нет неживых переходов.

$$\forall_{t \in T} \exists_{M,M'} i \xrightarrow{*} M \xrightarrow{t} M'$$

Рассмотрим примеры нарушения условий корректности WF-сетей.

- Для WF-сети , изображённой на **Рис.** 2(a) нарушено первое условие корректности. Срабатывание любого из переходов t1,t2 или t3 приводит к маркировке, из которой маркировка o недостижима. При маркировке c1 мы имеем бесконечный цикл,а в случае маркировки c2 или c3 переход t5 никогда не сработает, т.к. требует по метке на каждой из входных позиций.

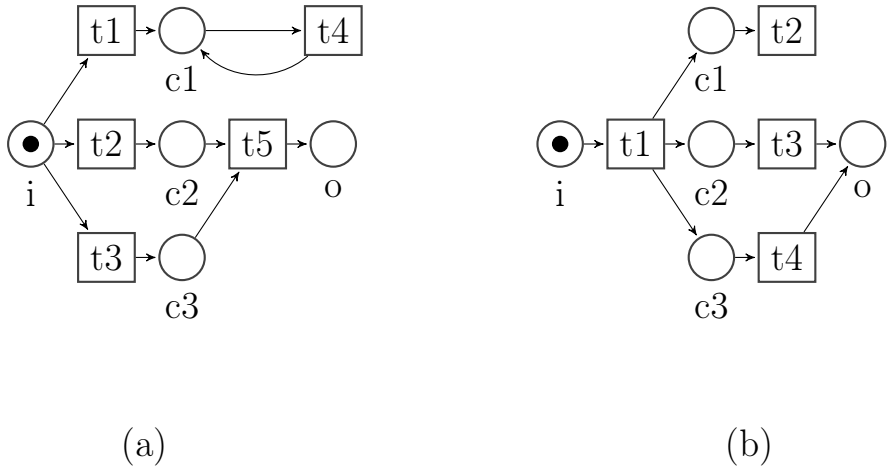


Рис. 2: This is a figure.

- В результате выполнения последовательности переходов $\langle t1, t3, t4\rangle$ WF-сети **Рис.** 2(b) нарушается второе условие корректности,

- В сети , изображённой на **Рис.** 2(b), так же нарушено третье условие корректности, т.к. переход t2 - неживой.

Для данной WF-сети PN = (S,T,F) мы хотим определить, является ли она корректной. В [2] показано, как свойство корректности соотносится с живостью и ограниченностью сети. Для того, чтобы связать эти понятия,дадим определение расширенной сети $\overline{PN} = (\overline{S}, \overline{T}, \overline{F})$, где $\overline{S} = S, \overline{T} = T \cup \{t^*\}, \overline{F} = F \cup \langle o, t^*\rangle, \langle t^*, i\rangle\}$.

Теорема 1. *WF-сеть PN корректна тогда и только тогда, когда ($\overline{PN}$,i) - живая и ограниченная маркировочная сеть.*

5.1 Струтурные характеристики корректности

Корректности WF-сети является её динамической характеристикой и из это вытекают некоторые проблемы:

- Для сложных WF-сетей задача определения корректности может быть весьма затратной (Для произвольных WF-сетей вычисление ограниченности и живости имеет экспоненциальную сложность[8]).

- Теорема 1 не определяет в какие именно компоненты WF-сети нарушают свойство корректности.

Поэтому полезно было бы знать какими структурными характеристиками обладают корректные WF-сети. Для этого будут рассмотрены некоторые классы WF-сетей: WF-сети со свободным выбором, хорошо структурированные WF-сети/

5.1.1 WF-сети со свободным выбором

Напомним определение свободного выбора в сети. Если для любых двух переходов $t_1, t_2 \in T$, для которых выполняется $\bullet t_1 \cap \bullet t_2 \neq \varnothing$, то необходимо чтобы $\bullet t_1 = \bullet t_2$, т.е. эти переходы являются частью одного OR-split.
Соблюдение правила свободного выбора не мешает моделировать параллельное,последовательное и циклическое выполнение задач.
Если же мы будем допускать сети не обладающие свойством свободного выбора,то тогда на выбор между конкурирующими задачами влияет очерёдность в которой выполнялись предшествующие задачи, что является недопустимым при грамотном проектировании. Нарушение свободы выбора,чаще всего возникает при сочетании распаралливания задач и маршрутизации с выбором.

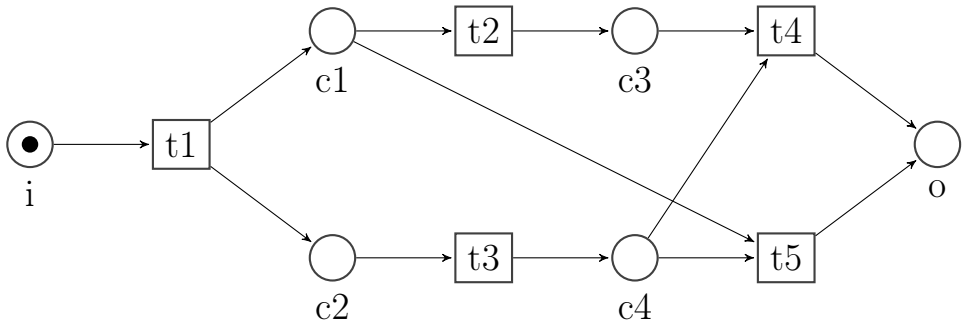


Рис. 3: This is a figure.

На **Рис.** 3 изображена такая ситуация. Сработавший переход t1 вводит параллелизм. Но тем не менее, в выбор между t2 и t5 ,т.к. переход t5 не возбуждён. Параллельное выполнение t2 и t3 приводит к ситуации, когда выполнить переход t5 невозможно. Тем не менее, если выполнение перехода t2 отложено до окончания выполнения t3, то тогда будет существовать выбор между t2 и t5. Но хотелось бы , чтобы параллелизм был отделён от выбора между альтернативами. Поэтому мы считаем конструкцию, приведённую на **Рис.** 3, неверной.

Вывод 1. *Для WF-сети со свободным выбором, задача проверки на корректность решается за полиномиальное время.*

Док-во. Пусть PN - WF-сеть со свободным выбором, тогда $\overline{PN}$ тоже будет обладать свободным выбором. В работе [10] доказано, что определение живости и ограниченности для($\overline{PN}$), можно за полиномиальное время. По Теорема 1 это эквивалентно корректности WF-сети.

5.1.2 Хорошо структурированные WF-сети

Другой подход для определения хорошей структурной характеристики workflow - это сбалансированность компонент типов AND/OR-split и AND/OR-join. Очевидно, что два параллельных потока инициированных AND-split, не должны объединяться через OR-join.Так же как и два альтернативных потока , инициированных компонентом OR-split не должны синхронизироваться через AND-join.

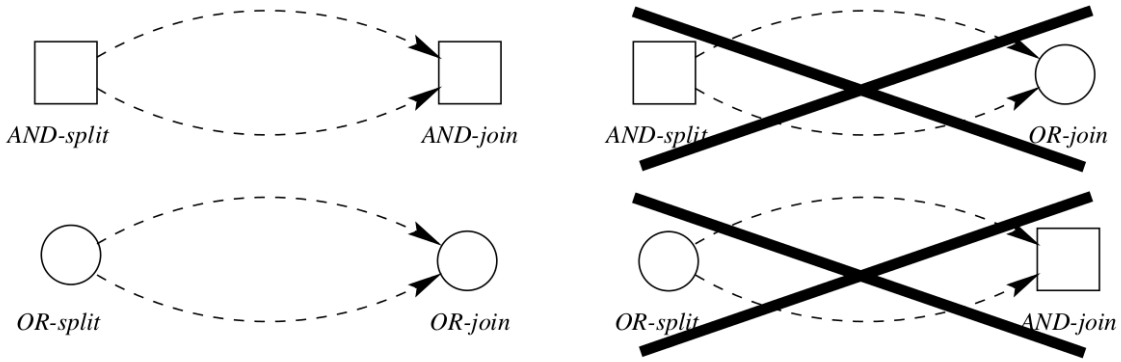


Рис. 4: Хорошие и плохие конструкции

Для того чтобы формализовать конструкции на рисунке **Рис.** 4 дадим следующее определение.

Хорошая организованность(Well-handled) *Сеть Петри PN называется хорошо организованной, если для любых пар вершин x и y таких , что одна из вершин является позицией , а другая переходом и для любой пары простых путей C_1 и C_2 из x в y,* $\alpha(C_1) \cap \alpha(C_2) = \{x, y\} \Rightarrow C_1 = C_2$.

Хорошая организованность может быть определена за полиномиальное время применяя алгоритмы нахождения максимального потока и минимального среза.

Хорошая структурированность(Well-structured) *WF-сеть является хорошая структурированной, если $\overline{PN}$ хорошая организованная.*

6 Преобразование графа состояний workflow к виду сетей Петри

Как уже говорилось в предыдущей главе, при моделировании динамических систем, таких как workflow, с помощью сетей Петри используется нотация, при которой состояния системы обозначаются позиции, а исполняемые задания - переходами.

По определению состояния workflow несёт информацию состоянии автомата и доступных сигналах на входных портах для каждого блока системы в определённый дискретный момент времени. Для каждого ребра графа состояний, по начальной и конечной вершине, мы можем определить, какие блоки и в каких состояниях асинхронно выполнили работу.

Каждое состояние workflow, представляется набором из одного или нескольких положений сети Петри, число которых равняется числу блоков, с сигналами на входных портах,и каждое положение - характеризует информацию о доступных сигналах на входных портах конкретного блока с конкретным состоянием автомата при данном состоянии workflow.

Соответственно, полученный граф состояний можно представить в виде сети Петри, путём сопоставления, каждому состоянию одной или нескольких позиций в контексте сетей Петри, а каждому ребру графа, соединяющему состояния, один и несколько переходов, соединяющих соответствующие позиции.

Рассмотрим более подробно правила сопоставления позиций сети Петри состояниям workflow. Мы хотим, чтобы каждый переход представлял работу одного блока за один шаг, следовательно позиция предшествующая переходу должна характеризовать готовность сигналов на входных портах блока. Таким образом каждое состояние workflow в контексте сетей Петри будет отображаться набор позиций,каждая из которых будет является группировкой доступных

сигналов для отдельного блока.

7 Построение стратегии запуска workflow в распределённой вычислительной среде

Задача построения стратегии запуска научно-инженерных workflow,как правило,включает в себя распределение заданий между доступными вычислительными ресурсами. Решение такой задачи обычно сводится к оптимизации функции затрат, зависящей от оценок времени работы заданий и производительности ресурсов. Более сложные планировщики, так же могут учитывать ещё и время ,затрачиваемое на передачу данных между задачами.
В силу того, что для нахождения оптимального решения задачи планирования должны быть рассмотрены все возможные варианты распределения заданий между ресурсами, задача планирования относится к классу NP-полных. Поэтому для нахождения около-оптимальных стратегии обычно используются эвристические подходы. Но получаемый результат в ходе применения таких подходов сильно зависит от точности используемых оценок времени работы заданий. Подходы к оценке времени выполнения могут быть разделены на три основные группы:

- Эмпирической модели,т.е. представляющие задания как чёрный ящик и предсказывающие их поведение на основе информации, полученной при прошлых запусках. В силу того, что большинство систем управления workflow способно получать такую информацию о времени работы запущенных заданий, то такой подход является потенциально успешным.

- Аналитические модели, пытающиеся описать производительность заданий в виде математических функций.

- Техники симуляции, выполняющие пробный запуск workflow на более простом наборе данных и использующие измеренные результаты в качестве начальной оценки.

Эвристические методы являются относительно новыми и быстро развивающимися методами. Среди этих методов выделяются эволюци-

онные и поведенческие (имитационные) методы. Поведенческие методы решения задачи глобальной безусловной оптимизации основаны на моделировании коллективного поведения самоорганизующихся живых или неживых систем. Взаимодействующие элементы этих систем, в общем случае, называются агентами. Ключевыми идеями поведенческих методов являются децентрализованность, взаимодействие агентов, простота поведения агентов. Известность получили следующие поведенческие методы решения задачи глобальной безусловной оптимизации: метод поведения пчёл; метод колонии муравьев; применяемый в данной работе метод роя частиц.

Планирование может осуществляться статически, т.е. когда схема работы workflow вычисляется до его запуска на основе имеющейся информации и строго соблюдается во время работы. При статическом планировании динамические изменения в вычислительной среде не учитываются. При динамическом планировании, в расчёт берётся как динамическая, так и статическая информация, и распределение ресурсов происходит во время работы.

В данной работе, будет предложен некоторый подход по статическому планированию сценариев workflow, соответствующих нашей модели. В частности, с помощью методов усреднения не детерминированных участков сети, будет построен ациклический граф задач, детерминировано задающий порядок запуска блоков.

7.1 Получение графа задач

Большинство существующих статического методов планирования применимы только для систем, представленных в виде ориентированного ациклического графа задач.

В предыдущей главе, мы показали, как можно представить динамику работу сценария workflow в виде сети Петри. Но полученная сеть Петри может содержать циклы и условные переходы. Вводя дополнительную эвристическую информацию о вероятностях переходов конечного автомата блока и оценки времени работы каждого блока, мы сможем перенести эту информацию и на построенную сеть Петри. Дальше в работе будут предложены методы преобразования недетерминированных участков сети Петри к детерминированному последовательному виду, сохраняя при этом временные характеристики.

Избавление от циклов

Data: Сеть Петри $PN^* = (S, T, F, P, W)$
,где

- $P : (S \times T) \bigcup (T \times S) \to [0, 1]$ - вероятности условных переходов.
  $\forall s \in S, t \in T, (t, s) \in F : P(t, s) = 1$ $\forall v1, v2 \in S \bigcup T, (v1, v2) \notin F : P(v1, v2) = 0$

- $W : T \to [0, +\infty)$ - время работы перехода.

Start:

1. Находим все простые циклы в сети PN;

2. Если какие-либо два цикла из найденных имеют общие вершины, то объединяем их в один.

3. **foreach** *Для каждого полученного цикла* **do**
   Определяем набор вершин цикла V;
   Находим Exits $= \{t : s \in S, t \notin V, (s, t) \in F\}$ - набор переходов не принадлежащих циклу,но смежных с положениями , входящими в цикл;
   $V^* = V \cup Exits$ - объединённый набор вершин; $M = \{P(v1, v2) : v1, v2 \in V^*\}$ - матрица вероятностей переходов между вершинами V^*;

   Считаем $M^* = M + M^2 + ... + M^n$ - транзитивное замыкание матрицы переходов. Считается с точностью $\max(M^n) > 0.01$. Элементу m_{kl} матрицы M^* соответствует мат.ожиданию числа прохождений метки через вершину l вершину, до выхода из цикла, при начальном положении в k.
   Находим позицию $s_{start} \in S \bigcap V$, самую близкую к начальной позиции i, являющаяся входом в цикл. Таким образом $m^* = m_{s_{start}}$ - строка матрицы M^*, соответствующая позиции s_{start};
   Добавляем в сеть новую позицию s_{end}, являющуюся завершающей позицией цикла.
   Определим Breaks $= \{t : t \in V, (t, s_{start}) \in F\}$ - набор переходов, принадлежащих циклу и соединения которых с s_{start} будут разорваны.
   foreach $t \in Breaks$ **do**
   Все рёбра $\{(t, s_{start})\}$, удаляются , и добавляются новые $\{(t, s_{end})\}$;

   end

   foreach $e \in Exits$ **do**
   Все рёбра $\{(v, e) : v \in V, (v, e) \in F\}$, удаляются , и добавляются новые $\{(v, e) : v \in V, (v, e) \in F\}$

   end

   foreach $e \in Exits$ **do**
   Все рёбра $\{(v, e) : v \in V, (v, e) \in F\}$, удаляются , и добавляются новые $\{(s_{end}, e)\}$. И $P_{new}(s_{end}, e) = m^*_e$

   end

   foreach $v \in V \bigcap T$ **do**
   Время работы перехода v умножается на матожидание числа запусков,т.е. m^*_v.
   $W_{new}(t) = W(t) * m^*_v$.

   end

   foreach $v \in V \bigcap S$ **do**
   Значения вероятностей условных переходов внутри цикла становится равным 1.
   $P_{new}(v, t) = P, ift \in T \bigcap V$

   end

   end

Алг. 1: представления цикла в сети Петри с известными вероятностями условных переходов в виде последовательности

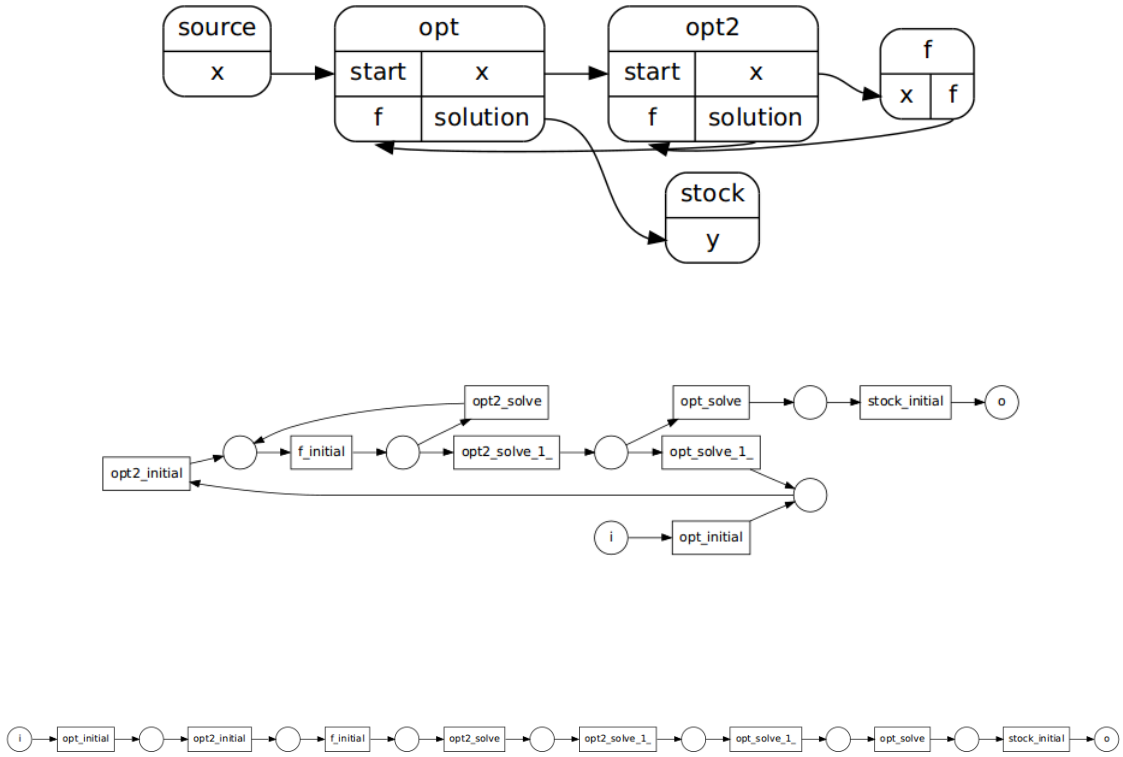
При при применении **Алг.** 1 к сети Петри с известными вероятностями условных переходов каждый цикл заменяется на после-

довательный участок сети, с пересчитанными значениями вероятностей выполнения условных переходов и усреднёнными временными затратами на выполнение переходов, содержавшихся в цикле.

Избавление от условных переходов

Любые два пути, с началом одной одной и той же начальной позицией типа OR-split и конечной позицией типа OR-JOIN можно привести к виду последовательного участка сети, в котором оба пути помещаются последовательно, а время работы каждого из переходов на этом участке умножается на вероятность условного перехода по этому пути.

Рис. 5: Пример схемы workflow с вложенным циклом, представление в виде сети Петри до и после процедуры устранения циклов



После того, как были проделаны процедуры по устранению циклов и условных переходов, сеть Петри будет представляться только последовательными и параллельными участками. Позиции при та-

кой структуре, могут быть соединены не более чем с двумя переходами. Удаляя все вершины графа, обозначающие позиции, и добавляя рёбра соединяющие смежные с этой позицией переходы, получим граф задач, в классическом его понимании. Так же для каждого задания мы знаем усреднённое время требуемое для выполнения.

7.2 Планирование на основе метода роя частиц

Задача планирования (или нахождения стратегии запуска) workflow, может быть сформулирована в виде:

"Найти такое статическое распределение блоков по узлам вычислительной сети из N допустимых узлов, что оценочное время выполнения всего workflow было минимальным при условии, что среднее значение показателя утилизации ресурсов было не меньше задаваемого уровня."

При этом для нашей модели работы workflow в распределённой вычислительной среде вводим следующие ограничения:

- Каждый блок запускается только на одном из доступных узлов;
- На каждом узле может быть запущено несколько блоков, но работать они могут только последовательно;
- Если на каком-то узле блоки не запускаются, то он считается узел считается неиспользуемым и не учитывается при вычислении уровня утилизации.

7.2.1 Вычисление времени работы workflow, при заданном распределении блоков между узлами

Функция вычисляющая время выполнения worklow(по построенному графу заданий),зависящая от распределения блоков по узлам вычислительной среды и будет целевой функцией, которую мы будем минимизировать. Алгоритм вычисления значений этой функции представлен на **??** 3.
Так же мы вводили условие на минимальный средний уровень утилизации вычислительных ресурсов. Под уровнем утилизации мы понимаем отношение эффективного времени работы к общему времени выполнения worklow. Средний уровень утилизации при использовании n узлов и полученном времени работы T может быть рассчитан

по формуле:

$$util(T, n) = T_{max}/(T * n) \tag{2}$$

T_{max} - суммарное время выполнения каждой задачи, что эквивалентно времени работы worklow при использовании 1 узла.

Data: B, N, G = (T,E),work, map, sched

- B = $\{b_1, ..., b_m\}$- набор из m блоков;
- N = $\{n_1, ..., n_k\}$ - набор из k доступных вычислительных узлов;
- G = (T,E) - граф задач для worklfow в котором, где T = $\{t_1, ..., t_l\}$ - набор задач, каждая из которых представляет работу блока в одном из его состояний, E представляет зависимости между запускаемыми задачами.
- work: $T \to [0, +\infty)$ - функция сопоставляющая каждой задаче время её выполнения;
- map: $T \to b, b \in B$- функция сопоставляющая каждой задаче блок , который она представляет;
- sched: $B \to n$ - функция сопоставляющая каждому блоку узел, на котором он будет работать.

Start:

1. timeTable - массив длиной k, каждый i-ый элемент - время завершения последней задачи, выполненной на узле $n_i \in N$. На шаге 0 массив состоит из нулей;
2. finished = {} - набор выполненных задач. На шаге 0 набор пуст;
3. ready = $\{t_{i_k}\}$ - набор задач, готовых начать выполнение. На шаге 0 состоит из задач, у которых нет зависимостей от других;
4. **while** *набор ready не пуст* **do**
   1. $t^* = argmin_{t_i \in ready}(timeTable[sched(map(t_i))] + work(t_i))$ - среди доступных задач выбираем ту, которая закончит своё выполнение раньше4 других;
   2. finished = finished $\cup\{t^*\}$; добавляем задачу к набору выполненных
   3. timeTable[$sched(map(t^*))$] += work(t_i); 4. next = $t : (t^*, t) \in E$; - находим задачи, которые зависят от t^*
   5. **foreach** $t_{next} \in next$ **do**
      pred = $t : (t, t_{next}) \in E$; - находим задачи, завершение которых необходимо для запуска t_{next}
      if *pred* $\subseteq$ *finished* **then**
      ready = ready $\cup\{t_{next}\}$;
      end
   end
   6. ready = ready $\setminus\{t^*\}$;
   end

Result: max(timeTable); возвращаем самое позднее время завершения заданий среди всех узлов

Алг. 2: вычисления времени выполнения worklfow, при фиксированном распределении блоков по узлам

7.2.2 Метод роя частиц

В методе оптимизации роем частиц (particle swarm optimization - PSO) агентами являются частицы в пространстве параметров задачи

оптимизации. В каждый момент времени (на каждой итерации) состояние частицы характеризуется координатами в пространстве решений (то есть, собственно, связанным с ними решением), а также вектором скорости перемещения. Оба этих параметра выбираются случайным образом на этапе инициализации. Кроме того, каждая частица хранит координаты лучшего из найденных ей решений, а также лучшее из пройденных всеми частицами решений – этим имитируется мгновенный обмен информацией между птицами.

На каждой итерации алгоритма направление и длина вектора скорости каждой из частиц изменяются в соответствие со сведениями о найденных оптимумах:

$$v_i = v_i + a_1 * rnd()(pbest_i - x_i) + a_2 * rnd() * (gbest_i - x_i) \qquad (3)$$

где v – вектор скорости частицы (v_i– его i-ая компонента), a_1, a_2 – постоянные ускорения, $pbest$– лучшая найденная частицей точка, $gbest$– лучшая точка из пройденных всеми частицами системы, – текущее положение частицы, а функция $rnd()$ возвращает случайное число от 0 до 1 включительно.

После вычисления направления вектора v, частица перемещается в точку $x = x+v$. В случае необходимости, обновляются значения лучших точек для каждой частицы и для всех частиц в целом. После этого цикл повторяется.

Целевая функция в нашей задаче задаётся на множестве из M*N элементов, где M - число блоков в схеме workflow,а N - число доступных вычислительных узлов. Но метод роя частиц работает в непрерывном пространстве. Тогда для того, чтобы их связать, пространство в котором движутся частицы будем считать M-мерным и непрерывным. А для отображения каждой из координат вектора этого пространство на множество из номеров доступных узлов введём отображение :

$$f(x) = \begin{cases} 1, & \text{если x}<\dfrac{1}{n}; \\ 2, & \text{если } \dfrac{1}{n} \le x < \dfrac{2}{n}; \\ \dots & \\ n, & \text{если } \dfrac{n-1}{n} \le x. \end{cases}$$

А для отображения всего вектора будем использовать:

$$F(\overline{x}) = (f(x_1), ..., f(x_M))$$

При инициализации частицы, каждая её координата будет задана случайным числом с равномерным на [0,1] распределением, а каждая координата скорости с равномерным на [-1, 1] распределением. При этом при перемещении частицы , будем следить, чтобы каждая её координата не выходила за пределы отрезка [0,1] , т.е.

$$x_i = \begin{cases} 0, & \text{если } x_i + v_i < 0; \\ x_i + v_i, & \text{если } 0 \leq x_i + v_i \leq 1; \\ 1, & \text{если } 1 < x_i + v_i; \end{cases}$$

На каждом шаге работы метода роя частиц, для каждой частицы считаем по Алг.2 время t_{new} работы workflow, используя распределении блоков по узлам, соответствующее дискретизации текущих координат частицы. Полученное значение целевой функции сравнивается с наилучшими для этой частицы и для всех частиц значениями, полученные на предыдущих шагах. После чего, если новое значение t_{new} при сравнении оказывается лучше, будет произведено обновление наилучших координат и значений. Используемый алгоритм сравнения представлен ниже:

```
if t_new < t_pbest and util(t, |F(x)|) ≥ util_min then
    pbest = x;
    t_pbest = t;
    if t_new < t_gbest then
        gbest = x;
        t_gbest = t;
    end
end
```

8 Заключение

В работе была разработана и формализована новая модель workflow, отличительной особенностью которой является наличие нетривиальных акторов, называемых в данной работе блоками, поведение которых описывается с помощью конечных недетерминированных автоматов. Так же был разработан алгоритм анализа этой модели, позволяющий находить возможные ошибки в проектируемом workflow на этапе построения. Результатом работы этого алгоритма является граф возможных состояний workflow.
Так же в работе предложен подход к статическому планированию запуска workflow в распределённой однородной среде на основе его графа возможных состояний. В рамках этого подхода предлагаются техники по построению сети Петри для графа состояний,и избавлению от условных переходов в сети Петри на основе дополнительно вводимой эвристической информации о времени работы блоков и вероятностях условных переходов. Для нахождения лучшего распределения блоков по узлам вычислительной среды применялась оптимизация методом роя частиц.
Модель workflow, алгоритмы её анализа и оптимизации, описанные в работе, были реализованы в виде программного модуля на языке программирования Python, используя который было смоделировано множество workflow, отвечающих как реальным инженерным задачам, так и абстрактным, и была проверена правильность разработанных алгоритмов и адекватность получаемых стратегий запуска.

Возможна дальнейшая работа по улучшению и расширению разработанной модели. В частности модель может быть расширена добавлением понятия иерархического workflow. Так же сложной и важной задачей является улучшение существующих и разработка новых техник планирования запуска в распределённой как однородной, так неоднородной среде.

Список литературы

[1]

[2] Y. Han, A. Sheth, C. Bussler, A Taxonomy of Adaptive Workflow Management, in: Conference on Computer–Supported Cooperative Work (CSCW-98), Seattle, WA, 1998. URL http://ccs.mit.edu/klein/cscw98/

[3] Grid-flow http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.93.5123

[4] Condor Project (www.cs.wisc.edu/condor).

[5] Symphony (www.zuni.cs.vt.edu/symphony).

[6] Cactus (www.cactuscode.org).

[7] UNICORE (www.unicore.sourceforge.net).

[8] Triana (www.triana.co.uk).

[9] Kepler (www.kepler.ecoinformatics.org).

[10] The Taverna Project (www.taverna.sourceforge.net).

10