МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра параллельных вычислений

Направление подготовки: 09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Образовательная программа: 09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

РАЗРАБОТКА И РЕАЛИЗАЦИЯ АЛГОРИТМОВ ВЫВОДА ОПТИМАЛЬНОГО ПЛАНА ТЕСТИРОВАНИЯ ФРАГМЕНТИРОВАННЫХ ПРОГРАММ

утверждена распоряжением проректора по учебной работе № 176 от «18» мая 2017 г.

Бедарев Николай Андреевич, группа 14204 _____

(подпись студента)

«К защите допущена»	**Руководитель ВКР**
Заведующий кафедрой,	д.т.н., профессор
д.т.н., профессор	заведующий кафедры ПВ ФИТ НГУ
Малышкин Виктор Эммануилович/……	Малышкин Виктор Эммануилович/……...
(подпись)	(подпись)
«……»……2018 г..	«……»……2018 г.

Дата защиты: «……»……20…г.

Новосибирск, 2018 г.

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра параллельных вычислений
Направление подготовки: 09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

УТВЕРЖДАЮ

Зав. кафедрой Малышкин Виктор Эммануилович
…….
(подпись)
«……»……20…г.

ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студент*у (ке)* Бедареву Николаю Андреевичу, группы 14204

Тема: Разработка и реализация алгоритмов вывода оптимального плана тестирования фрагментированных программ

утверждена распоряжением проректора по учебной работе от 18.05.2017 № 176

Срок сдачи студентом готовой работы……20.. г.

Исходные данные (или цель работы): создание инструмента, позволяющего применять математический аппарат вычислительных моделей из теории структурного синтеза программ для автоматизации рутинных действий.

Структурные части работы: обзор существующих систем автоматизации и накопления знаний, проектирование системы, реализация системы

Руководитель ВКР
заведующий кафедры ПВ ФИТ НГУ,
д.т.н., профессор
Малышкин Виктор Эммануилович/……..
(подпись)
«…»……20…г.

Задание принял к исполнению
Бедарев Николай Андреевич/……..
(ФИО студента) / (подпись)
«…»……20…г.

СОДЕРЖАНИЕ

Определения, обозначения и сокращения 4

Введение 6

Описание проблемы 6

Существующие подходы 6

Системы автоматизации выполнения команд 6

Системы накопления знаний 8

Вычислительные модели 9

Цель работы 10

Требования к системе 10

Требования к способу описания 10

Требования к интерпретатору 10

Задачи 11

1 Проектирование системы 12

1.1 Предлагаемое решение 12

1.1.1 Синтаксис вычислительных моделей 12

1.1.2 Исполнители 14

1.1.3 Принцип работы системы 15

1.1.4 Формализация нефункциональных свойств 16

1.1.5 Целевая функция. Алгоритмы работы 17

1.2 Характеристика предлагаемого решения 19

2 Реализация системы 20

2.1 Формат вычислительной модели 20

2.1.1 Метки 20

2.1.2 Циклы 21

2.1.3 Определения 22

2.1.4 Включения 22

2.2 Реализация интерпретатора 23

2.2.1 Исполнители 23

2.2.2 Вес операции 24

2.2.3 Схема работы интерпретатора 24

2.3 Использование 25

2.4 Тестирование 25

Заключение 27

Список использованных источников и литературы 29

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Вычислительная модель (ВМ) — это двудольный ориентированный граф. Одну долю назовем множеством переменных, другую долю — множеством операций.

Назовем две переменных **информационно зависимыми**, если есть путь, соединяющий их в графе.

ВМ можно также рассматривать как множество кортежей, каждый из которых соответствует одной операции, и включает в себя два подмножества переменных, один из которых определяет входящие дуги, а другой — исходящие. Переменные из этих подмножеств будем называть, соответственно, **входными** и **выходными** для данной операции. Говорят, что операция **вычисляет** свои выходные переменные из входных.

На практике каждая операция соответствует атомарному (с точки зрения ВМ) действию, которое может быть выполнено, а переменные обозначают те или иные промежуточные или конечные результаты описываемого процесса, которые могут быть уже получены (**переменная вычислена**) или ещё нет (**переменная не вычислена**). Для выполнения операции необходимо, чтобы все ее входные переменные были вычислены, а успешное выполнение операции влечет вычисление ее выходных переменных. Выполнение операции может оказаться и неуспешным, в этом случае вычисленность выходных переменных не изменяется.

Сценарий — это частично упорядоченное подмножество операций и подмножество переменных, называемых входными переменными сценария. Порядок на подмножестве операций должен следовать информационным зависимостям, т.е. для любой операции в сценарии все ее входные переменные либо являются входными переменными сценария, либо вычисляются как минимум одной из операций, предшествующих данной в этом порядке.

Будем говорить, что сценарий вычисляет множество переменных x из множества переменных y, если x — это множество входных переменных сценария, а каждая переменная из y либо принадлежат множеству входных переменных сценария, либо являются выходной переменной как минимум одной из операций сценария.

Атрибут операций — некоторая функция, которая отображает множество операций на множество вещественных чисел.

Атрибуты отражают нефункциональные свойства операций. Для каждой операции может быть определено несколько атрибутов, отражающих, например, оценочный расход времени на выполнение или требуемой памяти на диске. Используя значения атрибутов, становится возможным анализировать полученный сценарий с точки зрения его нефункциональных свойств.

Целевая функция — отображение множества сценариев на множество вещественных чисел.

Целевая функция задает критерий оптимизации, что позволяет определить **оптимальный сценарий** — сценарий, для которого целевая функция достигает минимума.

ВВЕДЕНИЕ

Описание проблемы

В современном мире большую актуальность набирает проблема накопления знаний и их последующего автоматического применения. При активном развитии информационных технологий появляется всё большее количество программного обеспечения и количество интерфейсов работы с данным программным обеспечением. Помимо различных интерфейсов работы программное обеспечение отличается различными нефункциональными характеристиками, в зависимости от которых, те или иные программы могут быть более предпочтительные в определенных обстоятельствах. Все это еще сильнее усложняет процесс эффективного автоматического использования знаний.

Особо остро эта проблема проявляется при использовании вычислительных кластеров для научного численного моделирования, которая сопровождается множеством вспомогательной работы, которая возлагается на исследователя. Примером такой работы может являться: загрузка исходного кода программы на вычислительный кластер, его последующая компиляция, установка необходимых вспомогательных библиотек и прочего окружения, загрузка данных для работы, создания скрипта запуска для системы очередей, и т.д. Все эти действия содержат обилие деталей, носящих технический характер и требующих от пользователя знаний, лежащих за пределами прикладной области исследователя. Как следствие вся эта рутина отнимает много сил и времени у пользователя.

В связи с тем, что это множество деталей поддается формализации целесообразно применить методы автоматизации, для снижения трудоемкости подобных рутинных действий.

Существующие подходы

Системы автоматизации выполнения команд

На сегодняшний день существуют средства и системы, позволяющие автоматизировать процессы сборки, тестирования и прочих сходных задач. К ним относятся, например, Automake/Autoconf[6], Qmake/Cmake[7,8], Scons[9], GNU

make[10], Apache Ant[11], MSBuild[12]. Характерным представителем подобных систем автоматизации является утилита GNU make.

GNU make – утилита, автоматизирующая процесс преобразования файлов из одной формы в другую. Чаще всего это компиляция исходного кода в объектные файлы и последующая компоновка в исполняемые файлы или библиотеки. Для работы утилиты к ней на вход подается файл, определенного формата, с описанием действий. Эти действия, как правило, разбиваются на некоторые последовательности действий, отвечающие на определённый этап работы. Примерами таких этапов могут быть:

- Скомпилировать исходные файлы в объектные файлы
- Скомпоновать исполняемый файл
- Удалить ненужные объектные файлы

Далее задается связь между этапами, т.е. какие этапы должны быть выполнены, прежде чем данный этап может быть исполнен. К примеру, для сборки исполняемого файла необходимо изначально собрать объектные файлы, а удалять объектные файлы можно только после сборки исполняемого файла.

Все прочие системы имеют схожие подходы к решению задачи автоматизации, с точностью до синтаксиса и дополнительных возможностей. Однако они все содержат общие недостатки:

- Отсутствует механизм описания различных способов получения искомого результата с их описанием нефункциональных свойств
- Отсутствует механизм работы с удаленными вычислителями, оценкой их доступности и загруженности

Описанные недостатки затрудняют описания избыточных решений, обеспечивающих достижение цели при невозможности выполнения отдельных операций или недоступности отдельных вычислительных мощностей, что при автоматизации данных действий дает возможность выбора оптимального, по некоторому критерию, сценария работы. В рассмотренных программных инструментах подобную логику приходится вкладывать вручную, что не позволяет в дальнейшем переиспользовать уже внесенные знания. Ситуация осложняется тем, что в данных системах акцент делается на решение конкретной задачи, возможно с разбиением ее на подзадачи.

Системы накопления знаний

В противовес системам автоматизации выполнения команд, одним из главных недостатков которых является отсутствие механизма описания нефункциональных свойств команд, ставятся системы накопления знаний. Системы накопления знаний позволяют накапливать знания об определенной предметной области, в зависимости от выбранного подхода накопления знаний.

Одним из самых распространенных подходов описания спецификаций некоторой предметной области и последующем их накоплением является использование онтологий. Одно из самых известных определений онтологии было дано в работе[3] Студера, а далее используется в работе[2]: “Онтология - это явная формальная спецификация разделяемой концептуализации”, где:

– под “концептуальной” моделью понимается абстрактная модель предметной области, описывающая систему понятий предметной области;

– под “разделяемой” моделью понимается согласованное понимание концептуальной модели определенной группой людей;

– “спецификация”, модель рассматривает описание системы понятий в явном виде;

– “формальная”, моделью рассматривается, что концептуальная модель является машиночитаемой;

– “явная” означает, что указан тип концептов, и ограничения на их использование явно определены.

В общем виде онтологии состоят из 4-х категорий элементов: аксиомы, отношения, понятия и экземпляры. Данная структура является достаточно универсальной, и мощной, что позволяет использовать данный подход во многих компьютерных системах, отвечающих, например, за информационных поиск, анализ текстов и т.д.. Однако, данный подход является крайне сложным в своем практическом применении. Основная сложность заключается в создании достаточно полной и непротиворечивой онтологии в определенной области. Также в дальнейшем эту онтологию будет необходимо использовать и расширять пользователям, что обязывает пользователя изучать довольно сложный способ описания. Помимо этого, следует отметить, что при добавлении новых отношений или правил вывода

онтология может изменяться сложно предсказуемым образом, что еще больше осложнит работу с ней.

Вычислительные модели

Для решения задачи автоматизации существует альтернативный подход, нежели в традиционных системах автоматизации выполнения команд. Предполагается, что для решения определенного спектра задач достаточно сформировать достаточно полную формальную машинно-ориентированную базу знаний о том, какие операции могут быть выполнены и как эти операции взаимосвязаны между собой. Это позволяет сводить задачу получения результата в этой предметной области к ее формальной спецификации, на основе которой формируется решение. Как следствие, данный подход позволяет накапливать знания в определённой предметной области абстрагируясь от конкретных прикладных задач, что в свою очередь, позволяет более широко переиспользовать накопленные знания, нежели традиционные подходы.

Для применения данного подхода предлагается использовать математический аппарат вычислительных моделей из теории структурного синтеза программ[1]. Вычислительная модель позволяет описывать знания об определенной предметной области в форме, допускающей автоматическое применение этих знаний компьютером[4]. Данный подход позволяет накапливать знания об определенной предметной области в виде атомарных шагов и их взаимосвязей, но в отличие от систем накопления знаний новые знания изменяют ее поведение хорошо предсказуемым образом. Как итог, задача выполнения рутинных действий сводится к заданию конечной цели, выраженной в множестве именованных переменных вычислительной модели.

Как следствие, вычислительные модели могут быть использованы для задач автоматизации рутинных действий пользователя при работе с вычислительными кластерами. В свою очередь, данный подход является мало изученным, и на данный момент отсутствует подходящий инструмент для работы с вычислительными моделями, который бы помог развивать данное направление.

Цель работы

Целью данной работы является создание инструмента, позволяющего применять математический аппарат вычислительных моделей из теории структурного синтеза программ для автоматизации рутинных действий.

Требования к системе

Создание подобного инструмента включает в себя 2 основных аспекта, а именно:

1. Необходимо разработать способ описания и последующего накопления знаний в виде вычислительных моделей.
2. Необходимо разработать интерпретатор, позволяющий работать с вычислительными моделями.

Требования к способу описания

Для разработки способа описания знаний необходимо разработать язык описания, и также разработать синтаксис описания данного языка. При этом на систему описания знаний накладываются следующие требования:

1. Простота описания вычислительных моделей, легкая читаемость. Простота в использовании данного инструмента является крайне важным требованием, в силу того, что с данным инструментом накопления знаний столкнутся все пользователи данной системы. Чем ниже сложность использования данного подхода, тем ниже порог вхождения новых пользователей, что в свою очередь поможет в продвижении данного подхода.
2. Расширяемость способа описания. Расширяемость способа описания является необходимым требованием, так как при дальнейшем изучении и развитии данного подхода инструментарий, скорее всего, будет необходимо дополнять и расширять.

Требования к интерпретатору

На интерпретатор вычислительных моделей также накладывают определенные требования:

1. Поддержка многовариантности получения результата. В силу изучаемого подхода, который позволяет конструировать сценарии получения результатов работы различными способами, необходимо, чтобы инструмент поддерживал данное ветвление и учитывал его при своей работе.
2. Заменяемый алгоритм построения сценария, который исполняется интерпретатором. В результате развития данного подхода, и расширения областей его применения может возникнуть необходимость в применении различных алгоритмов работы системы, обладающий необходимыми свойствами, специфичными для решения поставленной задачи. Как следствие, необходимо предусмотреть механизм, позволяющий встраивать различные алгоритмы работы, по мере необходимости.

Задачи

Для реализации данного инструмента были сформулированы следующие задачи:

1. Проектирование системы. Разработка синтаксиса языка описания вычислительных моделей.
2. Разработка алгоритмов работы системы.
3. Реализация системы – интерпретатора вычислительных моделей.
4. Тестирование работоспособности системы.

1 Проектирование системы

1.1 Предлагаемое решение

Для развития данного подхода необходимо расширить понятие вычислительных моделей, это было сделано в работе Прокопьевой А.В. [13] с помощью введения понятия атрибутов операций. Для задания атрибутов операций предлагается использовать синтаксическую конструкцию меток. Также в данной работе был разработан язык описания вычислительных моделей. На основе данного языка необходимо разработать синтаксис языка описания вычислительных моделей.

Помимо произведенных расширений языка, необходимо указать, что мы будем понимать под переменными ВМ. Под каждой переменной ВМ мы будем понимать **именованное состояние**, которое задает состояние какого-либо автоматизируемого процесса. Состояние будет считать достигнутым, если соответствующая ей переменная ВМ будет вычислена, и не достигнутым в противном случае.

Далее необходимо разработать интерпретатор вычислительных моделей, включающий в себя алгоритм работы, позволяющий на основе описания ВМ динамически конструировать эффективные сценарии и их исполнять.

1.1.1 Синтаксис вычислительных моделей

Важную роль в развитии данного подхода занимает разработка синтаксиса описания вычислительных моделей. Так как с данным синтаксисом столкнутся большинство пользователей, то данный синтаксис должен отличаться простотой описания, чтобы понизить порог вхождения новых пользователей. Помимо простоты описания немаловажным критерием является информативность записи, так как ВМ представляет из себя двудольный ориентированный граф, то эта информация должна быть отображена в способе задания ВМ. Также не стоит забывать о том, что при дальнейшем изучении и развитии данного подхода, скорее всего, возникнет необходимость в расширении данного синтаксиса для введения новых понятий и дополнительного функционала, то есть необходима расширяемость описания. Помимо этого, необходимо поддерживать масштабируемость описания ВМ, так как размер ВМ, описывающие конкретные предметные области могут быть крайне большими.

Исходя из вышеизложенных требований предлагается следующее:

Описывать ВМ отталкиваясь от описания отдельных операций, входящих в ВМ. Данный подход позволит достаточно просто описывать ВМ, исходя из того, что пользователь описывает отдельные операции и связи между ними. Где каждая операция характеризуется:

1. Множеством входных переменных, т.е. переменных, которые должны быть вычислены, прежде чем можно выполнять операцию.
2. Множество выходных переменных, т.е. переменных, которые будут считать вычисленными при корректном исполнении операции.
3. Множество меток.

Метки представляют из себя пару: название и значение. Метки служат для описания множества нефункциональных или иных свойств операций. Данный подход является расширяемым и позволит задавать не только предусмотренные свойства операций, но и вводить новые свойства, по мере дальнейшего развития проекта.

При достаточно полном описании какой-либо предметной области размер ВМ может оказаться крайне большим, поэтому необходимо ввести возможность разбивать описание на множество модулей, и дать возможность эти модулям взаимодействовать между собой. Для этой цели предлагается использовать механизм включений (include), который бы позволил включать логику одних ВМ в другие, а также механизм макроопределения (define). При этом, при использовании включений необходимо поддерживать возможность задания определений в используемых ВМ. В результате данный подход позволит создавать модульные параметризованные вычислительные модели.

Как было сказано выше, размер ВМ описывающей определенную предметную область может быть крайне большим, это может быть связано с необходимостью описания множества однотипных действий. В качестве механизма, позволяющего сократить запись однотипных операций предлагается использовать механизм циклов (for), который бы позволил сократить описание однотипных операций до единого, параметризованного цикла.

1.1.2 Исполнители

В зависимости от класса задачи и специфики операций, описанных в вычислительных моделях, способы исполнения и контроля за операциями должны различаться. Примером такого класса задач может являться выполнение команд на удаленном вычислителе, либо взаимодействие с системой очередей на вычислительном кластере. Также необходимо учитывать проблему балансировки нагрузки на вычислители и связанные с этим ограничения. Примером таких ограничений может являться ограничение на максимальное количество задач в очереди на вычисления, для конкретного пользователя, на вычислительном высокопроизводительном кластере. Как следствие, необходим механизм, позволяющий применять различные стратегии исполнения операций в зависимости от поставленной задачи.

Для решения данной задачи было решено разработать абстракцию исполнителя, основной целью которого является производить исполнение и контроль за исполнением операций. Данный подход дает возможность реализовывать своего исполнителя для каждого нового класса задач.

Инкапсулируя исполнителей данным образом, интерпретатор, тем не менее, должен обладать достаточной информацией о выполнении операций для эффективной своей работы. В качестве такой информации, которую будут предоставлять исполнители, предлагается использовать следующие характеристики:

1. Коэффициент производительности вычислителя (обозначим его как k).
2. Коэффициент дополнительной нагрузки на вычисления (обозначим его как m).

Данные коэффициенты позволяют учитывать изменение нефункциональных свойств операций, в зависимости от свойств исполнителей, на которых они исполняются. К примеру коэффициент k позволяет учитывать тот факт, что задачи, запущенные на более производительном вычислителе, будут исполняться быстрей, чем на менее производительном вычислители. А коэффициент m позволяет задавать дополнительную нагрузку на вычисления, которые могут возникнуть в связи с дополнительными действиями, которые необходимо будет сделать, для выполнения операции на данном вычислителе.

3. Вероятность успешного завершения выполнения операции и получения результата ее работы (обозначим ее как p).

Необходимость учитывать данную вероятность возникает при работе с удаленными вычислителями, где при исполнении операции могут возникать ситуации, когда мы не можем узнать результат выполнения данной операции, по причине недоступности вычислителя, например, из-за разрыва соединения. Аналогично, при использовании вычислительных кластеров может применять политика, когда при появлении крайне важных задач происходит “сброс” выполнения уже выделяющихся задач. Как следствие, целесообразно выбирать удаленные вычислители с более качественным соединением, и вычислительные кластера с меньшей вероятностью “сброса” задачи.

4. Наименование вычислителя.

Заданный интерфейс вычислителя позволяет задавать различные политики работы с вычислителями. Однако для работы с несколькими вычислителями, обладающими идентичными свойствами, необходимо их различать между собой. За это отвечает наименование вычислителей, которое в первую очередь направлено на реализацию возможность балансировки нагрузки на вычислители.

1.1.3 Принцип работы системы

В силу того, что данный подход позволяет задавать изменяющиеся во времени коэффициенты производительности и доступности отдельных вычислителей, интерпретатор в своей работе должен это учитывать. Помимо этого, необходимо учитывать тот факт, что некоторые операции, которые описаны в ВМ могут быть не выполнены в данных обстоятельствах, и при ошибке выполнения необходимо продолжить свою работу, учитываю ранее вычисленные переменные ВМ.

В качестве стратегии работы предлагается использовать итеративный метод работы, в котором:

1. В начале работы строится сценарий, на данную итерацию. Итерация заключается в исполнении всех операций, включенных в этот сценарий.
2. При успешном выполнении всех операций сценария работа интерпретатора считается корректно завершенной. При некорректном завершении какой-либо операции построенного сценария начинается построение нового сценария, учитывая уже достигнутые состояния.

Использования данного подхода позволяет на стадии планирования, во время конструирования сценария, учитывать множество, возможно, изменяющихся во времени параметров системы, однако данный подход не позволяет вносить изменения в уже сконструированный сценарий, по крайней мере до некорректного завершения операции содержащейся в сценарии, более подробно это ограничение рассматривается ниже.

1.1.4 Формализация нефункциональных свойств

На основе множества меток, сопоставленных каждой операции, в ВМ можно сделать вывод о нефункциональных свойствах операций и в дальнейшем, на основе данной информации, конструировать оптимальный сценарий. Однако, делать вывод о приоритете тех или иных операций, на основе множества нефункциональных свойств операций является сложной задачей. Для решения данной задачи предлагается применить распространённый способ, который подходит и в данном случае, а именно: свести множество нефункциональных свойств операций к одному окончательному весу, на основе которого и производить вычисление целевой функции. Помимо использования свойств операций необходимо учитывать знания о вычислителях, полученные от соответствующих исполнителей.

Для данной цели была разработан абстракция “весов” – объектов, которые производят отображение множества меток на множество вещественных чисел. Далее необходимо на основе данного числа и коэффициентов соответствующего исполнителя получить окончательный вес операции. Для этой цели предлагается использовать формулу:

$$F(k, m, p, w) = (w/k + m)/p$$

Где:

- w – вес операции, полученный с помощью “весов”.
- k – коэффициента производительности вычислителя.
- m – коэффициент дополнительной нагрузки на вычисления.
- p – вероятности успешного завершения выполнения команды.

Коэффициенты k, m и p вычисляются сответствующими исполнителями, на момент построения сценария.

1.1.5 Целевая функция. Алгоритмы работы

После формализации нефункциональных свойств операций необходимо на их основе разработать целевую функцию. Так как в зависимости от задачи, возможно, будут необходимы различные подходы к реализации целевой функции было решено разработать интерфейс, отвечающий за реализацию целевой функции. Целью данного интерфейса является выявление множества операций, которые необходимо выполнить для вычисления требуемого результата.

Далее перед системой возникает оптимизационная задача – выбор необходимого сценария из всех возможных. Данную задачу необходимо решать различными способами, в зависимости от свойств задачи и обстоятельств, в которых ее придется решать. Для настоящей исследовательской работы было решено разработать два алгоритма, отвечающие различным требованиям:

1. Переборный алгоритм, выбирающий оптимальный вариант из возможных.
2. “Быстрый” алгоритм, на основе алгоритма Дейкстры[5], находящий оптимальный сценарий работы в большинстве случаев.

Данные алгоритмы обладают широким покрытием задач, с которыми могут столкнуться пользователи. Однако, поддерживается заменяемость алгоритмов “решателей” и при необходимости можно разработать более подходящий алгоритм работы и встроить его в систему.

Схема работы переборного алгоритма:

Для простоты описания введем понятие ветви – пара множеств, множество операций и множество состояний. Вес ветви – это сумма весов всех операций принадлежащих данной ветви.

1. Выбираются все достигнутые состояния и формируются все возможные комбинации состояний. На их основе формируем ветви исполнения, для которых множество операций пустое.
2. Выбираются все операции, у которых отсутствуют входные состояния. На их основе формируем ветви исполнения, для которых множество состояний равно объединению множеств выходных состояний, для всех операций, включенных в ветвь.
3. Запускается алгоритм “деления” ветвей. Если ВМ содержит n (где n >= 1), операций, для которых все входные состояния принадлежат ветви V, то

образуются новые n ветвей, на основе ветви V, в которые включаются все состояния и операции из V, плюс одна из n соответствующих операций, а также все её выходные состояния.

4. Ветвь прекращает свое деление, если:
    a. Вес ветви стал больше суммарного веса всей ВМ
    b. В ветви содержаться все искомые выходные состояния
    c. Вес ветви больше одной из ветвей, которая содержит все искомые выходные состояния.
5. Как только образование новых веток невозможно, алгоритм завершает свою работу. Среди всех ветвей, содержащих все искомые выходные состояния выбирается ветвь с наименьшим весом.

Следует отметить, что данный алгоритм работает при условии, что все операции в ВМ имеют вес строго больше нуля. А также требует доработки, в случае использования рекурсивных вычислительных моделей.

Схема работы “быстрого” алгоритма:

1. На основе двудольного ориентированного графа ВМ строится ориентированный взвешенный граф состояний:
    a. где дуга u соединяется вершины v и w, если в графе ВМ существует операция у которой переменная соответствующая состоянию v принадлежит множеству входных состояний, а переменная соответствующая состоянию w принадлежит множеству выходных состояний.
    b. Вес дуги определяется с помощью функции F(k, m, p, w), определяющий конечный вес операции на данный момент.
2. Далее с помощью алгоритма Дейкстры находим кратчайшие пути от вершин, задающих результирующее состояния, до всех необходимых уже достигнутых входных состояний. Если какая-либо операция обладает больше чем одним входным состоянием, то ищется минимальный путь до уже достигнутых состояний для каждого входного состояния. Далее эти пути объединяются в единый сценарий, исключая повторы.

1.2 Характеристика предлагаемого решения

Дадим характеристику предлагаемому решению, с точки зрения поставленных требований.

Разработанный синтаксис языка описания вычислительных моделей позволяет компактно и достаточно просто описывать довольно большие вычислительные модели. Были разработаны несколько синтаксических конструкций, позволяющих упрощать описания вычислительных моделей, а также разбивать логику больших вычислительных моделей на множество более мелких. Данный подход позволяет сохранять простоту описания вычислительных моделей и помогают поддерживать масштабируемость их описания.

Предлагаемая архитектура интерпретатора вычислительных моделей позволяет использовать вычислительные модели в широком круге задач. Разработанная архитектура состоит из набора заменяемых модулей, которые можно разрабатывать и комбинировать в зависимости от специфики и требований определенной предметной области. В частности, заменяемый алгоритм вывода оптимального сценария позволяет выбирать алгоритм, обладающих необходимыми свойства для решения поставленных перед интерпретатором задач. В сумме с заменяемыми объектами реализующие интерфейс “весов”, позволяющих регулировать формализацию нефункциональных свойств операций, данный подход позволяет достаточно полно охватывать большую часть задач, с которыми может столкнуться пользователь.

Общий подход к управлению системы имеет итеративный характер, что позволяет достаточно просто ею управлять. В работе предлагается запускать новую итерацию работы только после некорректного выполнения одной из операций, что не позволяет, в некоторых случаях, в полной мере учитывать динамику изменения производительности и доступности вычислителей, однако данное ограничение не принципиальное, и есть возможность перестраивать сценарий выполнения либо после исполнения каждой операции, либо после получения сигнала об изменении параметров исполнителей.

2 Реализация системы

В рамках данной работы был разработан синтаксис предложенного языка описания вычислительных моделей. Для этих целей был разработан формат файла “cm”.

На основе разработанной архитектуры был реализован интерпретатор вычислительных моделей. В качестве заменяемых модулей были разработаны два вида исполнителей и два класса “весов”.

2.1 Формат вычислительной модели

В рамках работы был разработан формат описания вычислительных моделей. Вычислительная модель описывается в файле с расширением “cm”. Для удобства развития языка и дальнейшего его развития было принято решение при описании модели задавать версию языка. На данный момент развивается версия 1.0.

Для указания версии языка, на котором описана ВМ, файл должен начинаться со строки:

#version <версия языка описания>

ВМ для упрощения чтения пользователя разбита на строки, в каждой строке описываются операции с обозначением входных и выходных состояний, а также меток. Каждая строка файла описания вычислительной модели имеет формат:

<входные состояния>;<выходные состояния>;<операция>#<метки>

Для того, чтобы ввод множеств различных понятий в разрабатываемый синтаксис не утяжелял восприятие и читаемость вычислительных моделей предлагается использовать схожий синтаксис для описания каждого нового объекта синтаксиса. Для этого предлагается использовать следующую синтаксическую конструкцию:

%{[<наименование синтаксической конструкции>][<параметр 1>][<параметр 2>][...]}

2.1.1 Метки

Метки имеют формат:

<название>=<параметр>

На данный момент метки позволяют задавать следующие свойства операций:

1. Установка корректного возвращаемого значения операции. Для этого используется метка “result”, параметром которого является целое число.
2. Установка соответствующего веса операции. На данный момент реализовано 2 типа весов, которые описываются метками “time” и “memory”. Параметрами этих меток также являются целые числа.
3. Задания файлов, которые необходимо удалить после завершения работы. В случае, если операция порождает выходные файлы, которые не нужны в дальнейшей работе и будут только захламлять систему, их можно пометить как мусор, и они будут удалены при завершении работы интерпретатора. За отметку “мусора” отвечает метка “rubbish”, параметром которого является строка состоящая из названий этих файлов.
4. Любое иное описание операции, которое может использовать какой-либо модуль интерпретатор. (Например, количество потоков, которые будут задействованы при выполнении операции)

2.1.2 Циклы

Циклы представляют из себя n строк (n >= 0), которые заключены между строками следующего формата:

Объявление цикла:

#for <наименование цикла> #in <<перечисление>>:

З**авершение цикла:**

#endfor

Где <<перечисление>> – перечисление подстановок, перечисленное через ','. Общее количество элементов перечисленных в <<перечисление>> равно m(m >= 1). С помощью данных перечислений, на основе каждой из n строк, будет сгенерировано m новых строк, в которых на месте конструкции: %{[for][<наименование цикла>]}, будут подставлены элементы перечисления. Если строка не имеет данной конструкции, то с ней не производится никаких изменений.

Благодаря тому, что циклу сопоставляется название, в виде <наименование цикла>, то есть возможность использования нескольких вложенных циклов. При наличии вложенных циклов, строка, обозначающая завершения цикла, означает завершение лишь цикла, который объявлен последним.

2.1.3 Определения

Для использования определений изначально их необходимо определить с помощью строки:

#define <имя определения> <значение>

Далее, для применения определения необходимо написать:

%{[define][<имя определения>]}

В результате, будет произведена подстановка данной конструкции на соответствующее ей <значение>.

2.1.4 Включения

Включения являются механизмом, позволяющие использовать прочие вычислительные модели, то есть вычислительные модели, описанные в других файлах. Для реализации данного свойства в систему была добавлена переменная, хранящая в себе множество путей, по которым необходимо искать файлы описания вычислительных моделей.

Для использования включений изначально необходимо указать наименование файла вычислительной модели, логика которой будет включена в данную вычислительную модель. Для этого служит строка:

#include <имя файла включения>

Для применения включений предназначена следующая синтаксическая конструкция:

%{[include][<имя файла включения>][<наименование состояния>][<d_1>=<v_1>][<d_2>=<v_2>][...]}

Следует отметить, что использовать данную конструкцию можно лишь при описании множества входных состояний операций. Если при построении сценария, данная конструкция окажется во множестве входных состояний одной из операции сценария, то это означает, что:

1. Будет запущена подзадача, которая соответствует запуску интерпретатора над ВМ с названием <имя файла включения> и в качестве выходного состояния равного <наименование состояния>.

2. При запуске данной подзадачи будет реализована логика, по которой будут произведены определения с именами <d_1>, <d_2> и т.д. со значениями <v_1>, <v_2> и т.д. соответственно.

В дальнейшем данная логика включений может иметь множество направлений развития, например реализация способа включения непосредственно логики ВМ , без необходимости запуска подзадачи.

2.2 Реализация интерпретатора

Была разработана система на основе языка Java, по причине её широкого распространения и кроссплатформенности. Для работы интерпретатора необходим интерпретатор Java SE 8. Также, некоторые модули интерпретатора, отвечающие за контроль исполнения операций, были выполнены на языке python.

2.2.1 Исполнители

Для достижения цели данной исследовательской работы было спроектировано и реализовано два исполнителя. Первый исполнитель направлен на исполнение операций с помощью интерпретатора с помощью интерпретатора командной строки. Целью данного исполнителя является исполнение команд на локальной вычислительной машине. Однако, при работе с вычислительными кластерами задача контроля исполнения операций становится несколько сложней. Это связано с тем, что исполнение операций связано с использованием системы очередей. Для решения данной задачи был разработан второй исполнитель, основной целью которого является выполнение операций по следующему алгоритму:

1. Открыть ssh сессию с вычислителем
2. Поставить задачу в очередь на исполнение. Запомнить PID процесса.
3. Дождаться завершения исполнения операции.
4. Получить результат исполнения операции.

Следует отметить, что при ожидании завершения выполнения операции исполнитель опрашивает вычислитель о завершении операции. Для того чтобы снизить дополнительную нагрузку на вычислители применяется политика увеличения времени между каждым последующим запросом, до установленного предела. При таком подходе мы достаточно быстро узнаём о завершении операций, которые

выполняются достаточно быстро, и не нагружаем систему постоянными запросами при выполнении операций, которые выполняются большой промежуток времени.

2.2.2 Вес операции

Архитектура интерпретатора предусматривает создание множества различных весов, которые можно создавать под определенные класс задач. Поэтому, реализовывать универсальный класс, реализующий интерфейс весов нет необходимости. В качестве примеров реализации данного подхода были разработаны два класса, реализующих интерфейс весов, определяющие вес операций на основании меток “time” и “memory” соответственно. Данные классы позволяют протестировать данный подход на применимость множества различных “весов”.

2.2.3 Схема работы интерпретатора

Интерпретатор работает по итеративной схеме. Общий ход итерации описывается следующим алгоритмом:

1. Строится граф зависимостей на основе связей, описанных в файлах ВМ
2. На основе входных параметров, включающих в себя множество входных состояний и множество выходных состояний, а также предыдущих запусков, если такие были, выделяется множество уже достигнутых состояний. На их основе выстраивается граф достижимых зависимостей, относительно множества выходных состояний.
3. Для всех команд включенных в граф достижимых зависимостей высчитывается вес, при помощи выбранных объектов “весов” и предложенной формулой F(k, m, p, w)
4. На основе выбранного алгоритма, реализующего целевую функцию, строится сценарий.
5. Производится запуск сценария, в случае если команды являются информационно независимыми, то допускается параллельное их выполнение.

Первая итерация происходит при запуске интерпретатора, последующие в случае невозможности выполнить предложенный сценарий. После получения всех выходных состояний, либо получение вывода о том, что хоть одно из них не может быть получено работа интерпретатора завершается. При завершении работы

производится сборка мусора, на основе соответствующих меток исполненных команд.

2.3 Использование

Использование интерпретатора предполагает использование исполняемого jar файла:

java -jar interpreter.jar run …

Применение: run [options]

Options:

1. -c, --core – Название весов, которые будут использоваться при высчитывании веса команды. По умолчанию “time”
2. --dir-environment – Для задания директорий, файлы, находящиеся в которых задают состояний, которые будут считаться достигнутыми при старте системы. По умолчанию “.”
3. --cm, --files – Файлы ВМ, на основе которых будет работать интерпретатор
4. -o, --out – Наименование результирующих состояний работы интерпретатора.
5. --start-states – Список дополнительных состояний, которые будут считаться достигнутыми при старте системы.

2.4 Тестирование

Первый этап тестирования нацелен на проверку работоспособности интерпретатора, а также на правильность реализации синтаксических конструкций.

Для проверки работоспособности была составлена ВМ состоящая из операций echo, имитирующих различные операции, к которым были сопоставлены различные нефункциональные свойства с помощью меток “time” и “memory”. Было составлено 2 различные траектории, содержащие по 3 операции. В 1-ой траектории у каждой операции были выставлены метки: time=10, memory=100, у 2-ой же наоборот, time=100, memory=10. В результате запуска интерпретатора с выбором различных весов были исполнены различные операции, что и ожидалось.

Для проверки синтаксических конструкций были разработаны 3 теста. Для проверки каждой из конструкций было решено реализовать по две ВМ, выполняющие одни и те же, но в одном случае с использованием синтаксических конструкций, а в другом случае нет. Все тесты показали, что синтаксические конструкции работают корректно, и не вносят изменений в логику работы соответствующих ВМ.

Следует отметить, что данный инструмент использовался в работе Прокопьевой А.В., что показывает возможность применения данного инструмента не только на “синтетических” тестах, но и на практике.

Проведенное тестирование показало, что реализованные синтаксические конструкции и система в целом работают корректно. Непредсказуемого поведения не было выявлено.

ЗАКЛЮЧЕНИЕ

В результате данной работы была достигнута цель в создании инструмента, позволяющего применять математический аппарат вычислительных моделей из теории структурного синтеза программ для автоматизации рутинных действий пользователей, была достигнута.

Был разработан синтаксис описания вычислительных моделей. Разработан и реализован интерпретатор вычислительных моделей. В качестве модулей интерпретатора были разработаны и реализованы модули исполнителей и весов. Разработаны алгоритмы вывода сценария, а также предложен подход, позволяющий заменять данный алгоритм в зависимости от поставленной задачи.

На защиту выносится:

1. Синтаксис описания вычислительных моделей
2. Архитектура интерпретатора и его составных модулей, в том числе и алгоритмы работы построения сценария.
3. Реализация интерпретатора и множества составных модулей.

Дальнейшее развитие проекта включается в себя:

1. Развитие синтаксиса описания вычислительных моделей.
2. Реализация данного подхода средствами иных языков, для возможности более широкого его применения.
3. Усовершенствование подхода, реализация более сложных алгоритмов построения сценариев и балансировки нагрузки.

Выпускная квалификационная работа выполнена мной самостоятельно и с соблюдением правил профессиональной этики. Все использованные в работе материалы и заимствованные принципиальные положения (концепции) из опубликованной научной литературы и других источников имеют ссылки на них. Я несу ответственность за приведенные данные и сделанные выводы.

Я ознакомлен с программой государственной итоговой аттестации, согласно которой обнаружение плагиата, фальсификации данных и ложного цитирования является основанием для не допуска к защите выпускной квалификационной работы и выставления оценки «неудовлетворительно».

Бедарев Николай Андреевич _____

ФИО студента Подпись студента

« ____ »_____20 __г.

(заполняется от руки)

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Вальковский В.А. Синтез параллельных программ и систем на вычислительных моделях / В.А.Вальковский, В.Э. Малышкин. — Новосибирск: Наука. Сиб. отд-ние, 1988. — 129 с.
2. Харитонова Я.А., Подвальный Е.С. Онтологии как средство формирования базы знаний по многоальтернативным системам // Вестник ВГТУ. 2014. №4. URL: https://cyberleninka.ru/article/n/ontologii-kak-sredstvo-formirovaniya-bazy-znaniy-po-mnogoalternativnym-sistemam (дата обращения: 20.04.2018).
3. Studer R. Knowledge engineering: Principles and methods / Studer R., Benjamins R., Fensel D. // Data and Knowledge Engineering. - 1998. - № 25. - P. 161 - 197.
4. Victor Malyshkin. Active Knowledge, LuNA and Literacy for Oncoming Centuries // LNCS, Vol. 9465. 2015. P. 292-303.
5. Dijkstra E. W. A note on two problems in connexion with graphs // Numer. Math — Springer Science+Business Media, 1959. — Vol. 1, Iss. 1. — P. 269–271.
6. Autoconf. URL: https://www.gnu.org/software/autoconf/ (дата обращения 20.04.2018)
7. QMake. URL: http://doc.qt.io/archives/qt-4.8/qmake-manual.html (дата обращения 20.04.2018)
8. CMake. URL: https://cmake.org/ (дата обращения 20.04.2018)
9. SCons. URL: https://scons.org/ (дата обращения 20.04.2018)
10. GNU Make. URL: https://www.gnu.org/software/make/ (дата обращения 20.04.2018)
11. Apache Ant. URL: https://ant.apache.org/ (дата обращения 20.04.2018)
12. MSBuild. URL: https://docs.microsoft.com/visualstudio/msbuild/msbuild (дата обращения 20.04.2018)
13. Материалы конференции. Бедарев Н.А., Прокопьева А.В. Автоматизация на базе вычислительных моделей оптимизации и выполнения рутинных действий пользователя с вычислительными кластерами: Материалы 56-й Международной научной студенческой конференции МНСК-2018: Информационные технологии. / Новосиб. гос. ун-т. Новосибирск, 2018. 5 с.