**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Национальный исследовательский университет «Высшая школа экономики»**

**Московский государственный институт электроники и математики**

**Департамент компьютерной инженерии**

**Исследование вычислительного процесса в системе с управлением потоком данных**

**Методические указания**

**к лабораторному практикуму**

**Москва 2022**

Составитель С.М. Салибекян

Вычислительная система, управляемая потоком данных: Метод. указания к Лабораторным работам по курсу «Математическое и имитационное моделирование» / Моск. ин-т электроники и математики Национального исследовательского университета «Высшая школа экономики»; Сост. С.М. Салибекян. М., 2022 – 113 с.

Библиогр.: 8 назв.

**Аннотация**

Методические указания призваны познакомить студентов с приёмами проектирования и моделирования вычислительных систем с управлением потоком данных (dataflow). Пособие даёт как теоретические знания (представление алгоритмов в виде графов, принципы работы вычислительных систем, управляемых потоком данных), так и знания практические (проектирование конкретной вычислительной системы с управлением потоком данных, моделирование системы с целью определения ее характеристик и оптимальных параметров). Предназначается для студентов бакалавриата и магистров.

ISBN

Оглавление

[Введение 5](#_Toc109679422)

[1. Основы data-flow парадигмы вычислительного процесса 11](#_Toc109679423)

[**1.1. Теоретические основы data-flow** 11](#_Toc109679424)

[**2.2 Основные типы вершин потокового графа** 16](#_Toc109679425)

[2 Объектно-атрибутная архитектура вычислительной системы 19](#_Toc109679426)

[**2.1 Описание объектно-атрибутной архитектуры** 19](#_Toc109679427)

[**2.2 ОА-язык программирования** 27](#_Toc109679428)

[**2.3 Организация параллельных вычислений в ОА вычислительной системе с ограниченными вычислительными ресурсами** 36](#_Toc109679429)

[**2.4 Трансляция языков на базе ОА вычислительной системы** 38](#_Toc109679430)

[**2.5. Другие вычислительные задачи, решаемые на базе ОА архитектуры** 46](#_Toc109679431)

[3 Среда объектно-атрибутного программирования и моделирования 62](#_Toc109679432)

[**3.1 Интерфейс среды программирования** 62](#_Toc109679433)

[**3.2 Имитационное моделирование с помощью среды программирования и моделирования** 66](#_Toc109679434)

[**3.3 Визуализация результатов имитационного моделирования** 70](#_Toc109679435)

[4. Практическая часть 71](#_Toc109679436)

[**4.1 Работа 1: Моделирование параллельного вычислительного процесса с альтернативными ветвями** 71](#_Toc109679437)

[**4.1.1 Задание для работы 1** 71](#_Toc109679438)

[**4.1.2 Варианты заданий для работы 1** 71](#_Toc109679439)

[**4.1.3 Этапы выполнения работы 1** 77](#_Toc109679440)

[**4.1.4 Пример выполнения работы 1** 78](#_Toc109679441)

[**4.2 Практическая работа 2: Синтаксического разбор конструкции языка высокого уровня** 93](#_Toc109679442)

[**4.2.1 Варианты заданий для работы 2** 94](#_Toc109679443)

[**4.2.2 Составление графа состояний распознающего автомата** 96](#_Toc109679444)

[**4.2.3 Создание ОА программы синтаксического анализатора** 97](#_Toc109679445)

[**4.2.4 Порядок запуска лексического анализатора** 101](#_Toc109679446)

[Приложение А. Список милликоманд ФУ «Потоковое АЛУ с плавающей точкой» и ФУ «консольный вывод» 104](#_Toc109679447)

[Приложение Б. Миллипрограмма расчета объёма усеченного конуса 106](#_Toc109679448)

[Приложение В. Миллипрограмма с настройками для моделирования 108](#_Toc109679449)

[Приложение Г. Список милликоманд ФУ «Синтаксический анализатор» (FULex) и «Список» (FUList) 111](#_Toc109679450)

[Приложение Д. АО-программа синтаксического анализатора 112](#_Toc109679451)

**Введение**

Парадигма вычислительного процесса, управляемого потоком данных (data-flow) является достаточно перспективной ввиду того, что она позволяет максимально эффективно производить распараллеливание вычислений, по сравнению с классической парадигмой вычислений потоком команд (control-flow). В нашем случае парадигмой понимается способ описания алгоритма и методика его выполнения. Классические вычислительные системы control-flow (системы с управлением потоком команд) [Орлов С. А., Цилькер Б. Я. Организация ЭВМ и систем: Учебник для вузов. 2-е изд. — СПб.: Питер, 2011. — 688 с. А.В.Павлов, Архитектура вычислительных систем – СПб: Университет ИТМО, 2016. – 86 с.] основываются на понятии вычислительной нити (треад, thread). Тред представляет собой совокупность последовательно выполняемых исполнительным устройством (ИУ) команд [Гафаров Ф.М. Г12 Параллельные вычисления: учеб. пособие / Ф.М. Гафаров, А.Ф. Галимянов. – Казань: Изд-во Казан. ун-та, 2018. – 149 с.]. Адрес выполняемой в настоящей момент команды находится в счетчике команд (Program Counter). Таким образом, вычислительная система (ВС) может выполнять только одну команду в один момент времени, т.к. в его составе находится только одно исполнительное устройство (в большинстве случает это арифметико-логическое устройство (АЛУ)). Ветвление вычислительного процесса реализуется с помощью команд безусловного и условного перехода. Однако современные компьютеры, от которых требуется большая производительность, могут ее обеспечить только с помощью распараллеливания выполнения команд. В control-flow ВС это обеспечивается с помощью определенных «костылей»: конвейер команд, предсказание переходов, суперскалярная и мультискалярная архитетуры, где применяется несколько ИУ, и т.д. Несколько ИУ, объеденных для обработки одного треда, назовем процессорным элементом. Функция процессорного элемента – последовательная выборка из памяти команд и их выполнение. Также применяется многопоточность вычислений (multithreading), когда в состав ВС входит несколько процессорных элементов. Как правило, многопоточность обеспечивают несколько процессорных ядер на одном процессорном кристалле или несколько параллельно работающих процессоров (многопроцессорные ВС). Однако все эти решения при реализации параллельных вычислений значительно усложняют ВС, замедляют ее работу, повышают стоимость и энергопотребление, затрудняют написание параллельных программ из-за необходимости синхронизации работы вычислительных нитей. Еще один недостаток control-flow – негибкость вычислительного процесса, работающего согласно строгой последовательности инструкций. В частности, значительно усложняется выполнение программы с альтернативными ветвями (это программа, которая может получить один и тот же результат различными способами; ветви активизируются в зависимости от поступивших входных данных). Еще одним недостатком control-flow парадигмы является жесткая привязка операндов к ячейке локальной оперативной памяти вычислительного узла, что затрудняет реализацию вычислений на распределенных ВС, которые состоят из нескольких вычислительных узлов с собственной локальной памятью.

ВС парадигмы data-flow [Milutinovic V, Trifunovic N, Salom J, Giorgi R. The guide to dataflow supercomputing. USA: Springer; 2015. https://proxylibrary.hse.ru:2084/content/pdf/10.1007%2F978-3-319-16229-4.pdf] представляет собой совокупность автономных по управлению исполнительных устройств, обменивающихся между собой данными, оформленными в виде токенов. Токен – это операнд, снабженный служебной информацией, по которой исполнительные устройства идентифицируют его и выбирают соответствующий способ его обработки. С помощью служебной информации ИУ идентифицирует данные, пришедшие к нему, и выбирает способ их обработки. Данные, относящиеся к одной операции, могут накапливается в ИУ до прихода полного комплекта. Исполнительное устройство начинают вычислительную работу, как только оно получит на вход в виде токенов все необходимые для осуществления вычислительной операции данные. Операнды в data-flow парадигме передаются между ИУ по одному. Единицей информационного обмена между ИУ является токен. Результат выполнения операции ИУ снабжает служебной информацией и выдает его другим ИУ для последующей обработки.

Отличие control-flow от data-flow заключается в акценте на средства управления вычислительным процессом. В первом случае это – команда, т.е. порядок выполнения вычислительных операций задается последовательностью команд, упорядоченных в виде вычислительной нити. Во втором случае, вычислением управляют появляющиеся во время вычислительного процесса данные: поступающие в виде токенов данные принимаются ИУ-ами, после поступления всех необходимых для выполнения операции данных исполнительное устройство снабжает результат вычислений служебной информацией и передает сформированный токен другим исполнительным устройствам для дальнейшей обработки. Т.е. информация, переданная на одни ИУ порождает результаты вычислений, которые передаются следующим ИУ, те их тоже обрабатывают, передают дальше и т.д.

Data-flow парадигма может применяется как на аппаратном, так и на программном уровне (организация вычислительного процесса программными средствами, языки программирования высокого уровня).

История практического применения начинается с 1975 года, когда профессором Массачусетского университета Джеком Деннисом была реализована первая в мире машина, работающая в данной парадигме. Однако уже к 90-м годам прошлого века интерес к ней угас, т.к. производительность универсальных data-flow вычислительных систем оказалась намного ниже, а стоимость намного ниже классических ВС. Причиной тому послужило то, что вычислительным ядром таких систем были классические процессоры, которые, по определению, не могут работать быстрее классических ВС; а оборудование, позволяющее работать ВС в data-flow стиле только ухудшало ее работу. Интерес же к data-flow снова возник после 2000-х годов с появлением дешевых и производительных программируемых логических интегральных схем (ПЛИС, FPGA). Data-flow система «вшивалась» в ПЛИС, и представляла собой специализированную ВС. Такие устройства чаще всего применяются в качестве препроцессора для сигнального процессора, выполняя предварительную обработку данных. Парадигма data-flow также нашла свое применение в языках программирования. Тут следует упомянуть акторное программирование [Г.И. Ревунков, Ю.Е. Гапанюк Введение в функциональное и логическое программирование / Электронное учебное издание. М.: МГТУ им. Н.Э. БАУМАНА, 2017 – 100 с., Федотов И.Е. Модели параллельного программирования. - М.: СОЛОН-ПРЕСС, 2012. - 384 с.], когда вся программа представляет собой набор автономных процедур, принимающих свои операнды по-отдельности в виде токенов. Результат вычисления актор выдает в виде токенов и пересылает его другим акторам-потребителям. К акторным языкам можно отнести LabView, Swift, Erlang [The official home of the Erlang Programming Language URL: <https://www.erlang.org/docs>], Scala [Мартин Одерски, Лекс Спун, Билл Веннерс

Scala. Профессиональное программирование . — СПб.: Питер, 2017] и т.д.

Data-flow ВС представляются весьма перспективными, ввиду отсутствия такого элемента как программный счетчик. Таким образом, никаких ограничений параллелизма, кроме коэффициента параллелизма реализуемого алгоритма, теоретически не существует. Правда обратной стороной этого преимущества является слишком большое потребление вычислительных ресурсов во время вычислительного процесса. Еще одни недостаток – невозможность применения кэш-памяти ввиду отсутствия локализации обращений к памяти, как в ВС control-flow, работающей в control-flow парадигме. Некоторые data-flow ВС также может подвергаться опасности информационного взрыва, когда информационный поток в ВС лавинообразно нарастает и ВС уже не в состоянии его обработать.

Одной из областей применения ОА ВС (как и других data-flow систем), является вычисление по алгоритму с альтернативными вычислительными ветвями. В таких алгоритмах есть несколько способов получения одного и того же результата. По какой вычислительной ветке первым будет получен результат, зависит от количества вычислительных операций, а также от наличия и последовательность прихода к ВС исходных данных для вычислений. Классическая (control-flow) ВС не оптимальна для подобных вычислений – на них они очень громоздки при больших количествах альтернатив.

Ввиду предполагаемой перспективности data-flow парадигмы весьма актуальной задачей является моделирование data-flow ВС с целью определения ее характеристик. Настоящее методическое пособие предназначено для освоения методики моделирования data-flow вычислительного процесса на примере алгоритма с альтернативными ветвями. Моделирование производится на примере вычислительной системы объектно-атрибутной (ОА) вычислительной системы (ВС) [Салибекян С.М., Панфилов П.Б. Объектно-атрибутная архитектура – новый подход к созданию объектных систем // Информационные технологии. 2012, №2 стр. 8-14 Салибекян С. М., Панфилов П. Б. Вопросы автоматно-сетевого моделирования вычислительных систем с управлением потоком данных // Информационные технологии и вычислительные системы. 2015. № 1. С. 3-9], относящейся к классу data-flow. Для управления данной ВС разработан специализированный язык программирования (ОА-язык), а также среда программирования и моделирования. В ней с помощью ОА-языка можно описать состав виртуальной ВС и запустить на ней эмуляцию вычислительного процесса. Средства имитационного моделирования позволяют получить характеристики этого вычислительного процесса.

Целью настоящего лабораторного практикума является отработка навыков моделирования вычислительных систем с целью оценки их характеристик и выбора оптимальных параметров, и отработка навыков анализа параллельного вычислительного процесса на базе парадигмы data-flow, а также навыков создания транслятора языка высокого уровня на базе data-flow.

1. **Основы data-flow парадигмы вычислительного процесса**

**1.1. Теоретические основы data-flow**

Можно сказать, что начало развития data-flow модели вычислений лежит в начале 60-х годов прошлого века [Ключев А.О., Кустарев П.В., Ковязина Д.Р., Петров Е.В. Программное обеспечение встроенных вычислительные систем. – СПб.: СПбГУ ИТМО, 2009. – 212 с.]. Необходимость разработки такой модели именно в то время объясняется появлением сложных параллельных и распределенных вычислительных систем, обрабатывающих сразу несколько потоков данных или осуществляющих другую параллельную обработку данных. Старые формальные модели того времени были слабо приспособлены для описания их работы. Первый формализм, способствующий решения этой проблемы (сети Петри) появился в 1962 году. С его помощью можно было описывать работы параллельных вычислительных систем, однако описание потоковых вычислений с ее помощью было достаточно громоздким. А в 1963 г. Естрином и Турном (Estrin and Turn) были предложены первые потоковые модели. В 1966 году Карп и Миллер (Karp and Miller) предложили потоковые графы без переходов [[R. Karp](https://www.semanticscholar.org/author/R.-Karp/47546648), [Raymond E. Miller](https://www.semanticscholar.org/author/Raymond-E.-Miller/1400120462) Properties of a model for parallel computations: determinacy // Siam Journal on Applied Mathematics, 1966]. В 1969 г. Родригесом произвел формализацию и расширение модели Эстрина. А в 1971-м году появился первый потоковый язык программирования, разработанный Чемберлен (Chamberlain). В 1974-м Каном (Kahn) предложил сети процессов с бесконечными очередями (процессные сети Кана – Kahn process network (KPN)). В этом же году Денисом (Dennis) создана потоковая модель с буферами на один элемент. Арвинд и Гостелов (Arvind and Gostelow), а также независимо Гард и Ватсон (Gurd and Watson) предложили потоковую модель с тегированными элементами. В 1975 году Йордон и Константайн (Yourdon and Constantine) предложили методику структурного проектирования программного обеспечения с декомпозицией системы на процессы и потоки данных.

Итак, приступим к описанию основ data-flow. Любой вычислительный процесс можно представить с помощью ориентированного графа, приписав его вершинам вычислительные операции (арифметические, логические и т.д.), а дугам - операнды (данные) этих операций. Такой граф называется потоковым. Например, операцию сложения можно представить так (рис. 1):

**+**

x1

x2

y

Рис. 1 Обозначение операции на потоковом графе

Входящие в вершину дуги – входные данные для операции, выходящая дуга – результат выполнения операции (в данном случае сложения), метка на узле обозначает вычислительную операцию. Точки входа данных в узел называются портами (port). Некоторые операции, например, вычитание, чувствительны к порядку следования операндов: например, если поменять местами вычитаемое и уменьшаемое, то результат операции будет другим. Поэтому у операций, чувствительных к перестановке операндов, порты и нумеруются: цифра 1 обозначает первый операнд, а цифра 2 – второй и т.д.. Также можно пометить названия операндов и результата (в нашем случае x1, x2 – операнды, y – результат). См. рис. 2.

**­–**

x1

x2

y

**1**

**2**

Входной порт

Порт

Входной порт

Выходной порт

Рис. 2 Обозначение операции на потоковом графе с пометками операндов и результата, а также номеров операндов

Результат выполнения операции подается в качестве операнда на порты одной или сразу нескольких операций. Таким образом можно описать любой алгоритм. Исходные данные для алгоритма на рисунке будем обозначать прямоугольниками, операции преобразования данных - гружком. Такой граф будем называть потоковым графом алгоритма. Например, арифметическое выражение y=ax2+bx+c посредством графовой модели можно представить так (рис. 3):

**A**

**B**

**C**

**x**

**+**

**+**

**\***

**\***

**\***

**Y**

Рис. 3 Потоковый граф арифметического выражения

При реализации алгоритма, представленного в виде потокового графа, есть два глобальных подхода: управление командами (control-flow) и управление данными (data-flow). Первый подход считается классическим – он используется в подавляющем большинстве современных компьютеров (фон Неймановская архитектура) [Архитектура ЭВМ и систем [Электронный ресурс] : учебное пособие / Ю.Ю. Громов, О.Г. Иванова, М.Ю. Серегин, М.А. Ивановский, В.Е. Дидрих. – Тамбов : Изд-во ФГБОУ ВПО «ТГТУ», 2012. – 200 с.]. Заключается он в том, что вычислительный процесс управляется потоком команд: процессор принимает коды, описывающие команды, интерпретирует их и выполняет. Получается, что для ВС во время вычислительного процесса принимает на входи и обрабатывает узлы потокового графа алгоритма. Так, типичная команда подобной ВС имеет следующий формат (рис. 4):

|  |  |  |  |
| --- | --- | --- | --- |
| КОП | Операнд 1 | Операнд 2 | Результат |

где КОП - код операции (код, который обозначает, какую операцию надо сделать);

первый операнд, второй операнд - указание, из каких ячеек памяти следует брать операнды;

результат - указание, куда следует поместить результат выполнения операции.

Рис. 4 Формат control-flow команды

Данная конструкция описывается узлом графа алгоритма, в который входит два операнда, и выходит один результат (рис. 5):

**+**

C

B

A

Где A, B – первый и второй операнды;

C – результат вычислений.

Рис. 5 Представление control-flow команды с помощью узла потокового графа

Таким образом, первое поле классической команды помечает узел потокового графа, а три других - дуги. Вычислительная нить (последовательность, выполняемых друг за другом, команд) «оплетает» потоковый граф алгоритма, обходя все его вершины. Для одного графа имеется множество возможных наложений вычислительной нити (рис. 6). Метод наложения нити на потоковый граф зависит от архитектуры ВС, выполняющей алгоритм, а также от программиста, написавшего control-flow программу. Для увеличения параллелизма вычислений в современных процессорах применяется несколько вычислительных нитей (multithreading), работающих параллельно. Каждая такая нить может выполняться отдельным процессором или процессорным ядром. Однако multithreading-гу свойственны существенные недостатки: он требует больших ресурсов для синхронизации вычислительных нитей, а также программисту необходима высокая квалификация, чтобы писать подобные приложения.

Вычислительная нить



Рис. 6. Наложение вычислительной нити на потоковый граф

Все операнды control-flow системы должны быть размещены в ячейках оперативной памяти. Т.к. дуга потокового графа есть описания передачи одного операнда, то адрес операнда в оперативной памяти выполняет роль метки дуги потокового графа, по которой ВС строит вычислительный процесс. Таким образом, метка операнда в виде его адреса в локальной оперативной памяти является неотъемлемой частью организации вычислительного процесса. Этот факт вызывает трудности при реализации control-flow вычислений на ВС с распределенной памятью, где память делится на несколько вычислительных сегментов с автономной памятью, а обмен данными между ними осуществляется с помощью сообщений по линиям связи [Стин ван М., Таненбаум Э. С. Распределенные системы / пер. с англ. В. А. Яроцкого. – М.: ДМК Пресс, 2021. – 584 с.].

В системах, управляемых данными [Jurij Silk, Borut Robic and Theo Ungerer «Asynchrony in parallel computing: From dataflow to multithreading» Institut Jozef Stefan, Technical Report CDS-97-4, September 1997. , Milutinovic V, Trifunovic N, Salom J, Giorgi R. The guide to dataflow supercomputing. USA: Springer; 2015. https://proxylibrary.hse.ru:2084/content/pdf/10.1007%2F978-3-319-16229-4.pdf], акцент делается не на описание команд, а на описание передачи данных, т.е. дуг потокового графа. Здесь кодируются не команды, а данные, которые передаются между операциями сначала в качестве результата вычислений, а потом в качестве операндов. Пересылка данных между узлами по дугам потокового графа осуществляется посредством токенов (токен представляет собой один операнд, снабженный служебной информацией).

**2.2 Основные типы вершин потокового графа**

Для того, чтобы описать любой алгоритм, в том числе с условными переходами и циклами, в потоковом графе необходимы вершины шести типов (рис. 7).

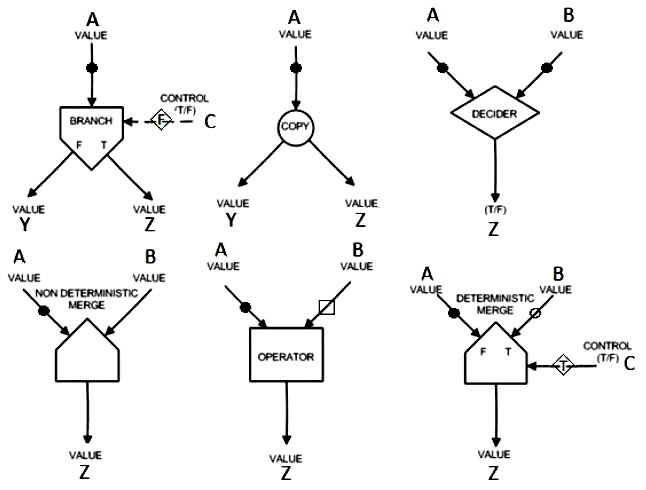


Рис. 7. Типы вершин потокового графа

Вершины типа «OPERATOR» выполняет вычислительную операцию над нескольким операндами, приходящими к ней на входные порты. Результат выполнения операции выдается через выходной порт. «COPY» осуществляет дублирование токена со входного порта на два выходных порта. «NON DETERMING MERGE» перенаправляет токены с нескольких входных каналов в один выходной. Следующие три типа предназначены для реализации условных переходов программы. «DESIDER» реализует предикат (т.е. функцию, возвращающую логическое значение). Токен с вычисленным логическим значением, поступает на управляющий порт узлов типов «BRANCH» или «DETERMING MERGE». Узел «BRANCH» передает входной токен на порт с пометкой «T», если на управляющей вход приходит токен со значение «правда» («true»), и на порт «F» иначе. «DETERMING MERGE» пропускает токен с порта «T», если на управляющей вход приходит токен со значение «правда» («true»), и с порта «F» иначе. Приведем потоковый граф алгоритма для решения квадратного управления (рис. 8.). В нем присутствует элемент «BRANCH», который пропускает вычисленный дискриминант далее по потоковому графу только в том случае, когда он больше или равен нулю (условие проверяется на узле типа «DESIDER»).



Рис. 8. Потоковый граф решения квадратного уравнения

**2 Объектно-атрибутная архитектура вычислительной системы**

В данной работе будет производиться проектирование объектно-атрибутной (ОА) ВС [Салибекян С.М., Панфилов П.Б. Объектно-атрибутная архитектура – новый подход к созданию объектных систем // Информационные технологии. 2012, №2 стр. 8-14 Салибекян С. М., Панфилов П. Б. Вопросы автоматно-сетевого моделирования вычислительных систем с управлением потоком данных // Информационные технологии и вычислительные системы. 2015. № 1. С. 3-9], относящейся к классу data-flow. ВС архитектуры ОА работает по следующим принципам.

**2.1 Описание объектно-атрибутной архитектуры**

Все данные, передаваемые между функциональными устройствами (ФУ), снабжены атрибутом, идентифицирующим эти данные. ФУ – это ИУ, выполняющее определенный набор операций над входными данными. Каждое ФУ в большинстве случаев ассоциируется с конкретным узлом потокового графа. Все ФУ (сумматоры, умножители, блоки работы с плавающей точкой и т.д.) объединены между собой шиной данных-атрибута (ШДА), по которой происходит передача токена (милликоманды). Архитектура ОА ВС представлена на рис. 6. В состав ФУ входят внутренние регистры (контекст ФУ) для хранения промежуточных данных для вычислений (так, в состав ФУ «Сумматор» будут входить три внутренних регистра: регистры 1-го и 2-го операндов и регистр результата вычисления). Атрибут милликоманды содержит два поля: первое - код ФУ, которому данные предназначаются; второе - описание данных, где указывается, как ФУ должно эти данные интерпретировать: например, для сумматора указывается, что данные являются 1-м или 2-м слагаемым (если необходимо произвести сложение) или вычитаемым, или уменьшаемым (если необходимо произвести вычитание) и т.д. ФУ, входящие в состав ВС, могут не только принимать и обрабатывать данные, но и выдавать на ШДА результат своей работы. Для этого в контекст ФУ включаются регистры, где хранятся атрибуты, прикрепляемые в качестве атрибута к результату работы ФУ: полученный результат вычислений снабжается соответствующим атрибутом и сформированный таким образом токен выдается на ШДА. Благодаря этому в ВС реализуется распределенное управление (т.е. нет единого блока, который централизованно осуществляет планирование вычислений в ВС. Это позволяет ВС осуществлять самораспараллеливание вычислений и самостоятельно выбирать наиболее оптимальный путь решения вычислительной задачи. Однако распределенное управление требует больших аппаратных затрат, ведь практически на каждую операцию нужно выделять отдельное ФУ. Так, для алгоритма вычисления квадратного многочлена, приведенного выше, потребуется пять ФУ: три умножителя и два сумматора.

ФУ 3

**…**

ФУ*m*

ФУ 2

ШДА

ФУ 1

Рис. 6. Архитектура объектно-атрибутной вычислительной системы

Описание каждой команды включает в себя атрибуты операндов, и атрибуты, которые необходимо «прикреплять» к результатам вычисления, когда они будут передаваться по ШДА. Перед началом работы ФУ следует запрограммировать. Так, для алгоритма вычисления квадратного трехчлена надо задействовать 5 ФУ: 3 умножителя и 2 сумматора.

**A**

**B**

*ж***м**

**C**

**x**

**+**

**+**

**\***

**\***

**\***

**Y**

*Умножитель 1*

*Умножитель 2*

*Умножитель 3*

**Сумматор 1**

*Сумматор 2*

*Выходной регистр*

Рис. 7. Распределение операций между исполнительными устройствами

Однако количество ФУ можно сократить. «Умножитель 2» и «Умножитель 3» всегда будут работать последовательно, т.к. между ними есть непосредственная связь по данным (когда результат вычисления одного ФУ передается в качестве операнда на другой ФУ). Так как эти устройства однотипны, две вычислительные операции можно произвести на одном ФУ без уменьшения времени выполнения вычислений. Так же в одном ФУ можно реализовать последовательные операции «Сумматор 1» и «Сумматор2». Таким образом, ВС для реализации данного алгоритма понадобятся всего три исполнительных устройства (рис 7).

**A**

**B**

**C**

**x**

**+**

**+**

**\***

**\***

**\***

**Y**

*Умножитель 1*

*Умножитель 2*

*Умножитель 2*

*Сумматор 1*

*Сумматор 1*

**1**

**2**

**4**

**4**

**4**

**5**

*л*

**6**

*о*

**7**

1

*а*

**8**

*ол*

**9**

*ма*

*Результат*

Рис. 7. Оптимальное распределение операций между исполнительными устройствами и индексация атрибутов токенов

Для программирования ВС каждому операнду, который станет передаваться по ШДА, следует присвоить свой уникальный идентификатор (атрибут), т.е., например, приписать номер каждой стрелке на потоковом графе программы. Внимание, индексируются не акты передачи операнда (дуги графа), а операнда. Обратите внимание, что на рис. 7 операнд «x» имеет индекс 4 – и индексом 4 помечаются все дуги, исходящие из узла с пометкой «x».

После можно составить таблицу программы для всех ФУ (таб. 1). В графе «Атрибуты операндов» перечисляются атрибуты (индексы) всех входящих дуг. Внимание, порядок операндов существенен для операций, чувствительных к перестановке операндов (например, вычитание или деление): в этом случае сначала записывается первый операнд, затем второй.

Таблица 1. Таблица операндов для программы вычисления арифметического выражения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наименование ФУ | Атрибуты операндов | Выполняемая операция | Комментарий | Ярлык выходных данных |
| Умножитель 1 | 2,4 | \* | B\*x | 5 |
| Умножитель 2 | 4,4 | \* | x2 | 6 |
|  | 1,6 | \* | A\* x2 | 7 |
| Сумматор 1 | 3,5 | + | B\*x+c | 8 |
|  | 7,8 | + | A\* x2+B\*x+c | 9 |

Для анализа работы ВС используется диаграмма Ганта [Калугин Владимир Евгеньевич Качество в проектировании и разработках: методические указания по выполнению курсового проекта [Электронный ресурс] / Е.В. Калугин, В.А. Осит – Омск: СибАДИ, 2017 – 40 с. URL: https://portal.sibadi.org/pluginfile.php/101557/mod\_resource/content/1/esd395.pdf , Рассел, Джесси Диаграмма Ганта / Джесси Рассел. – М.: VSD, 2012. – 591 c.], которая служит для визуализации поведения параллельно работающих вычислительных устройств во времени. Графики работы каждого ФУ располагаются друг под другом. А по оси "x" откладывается время. Пример приведен на рисунке 8. Передача данных между ФУ обозначается стрелкой. В том случае, когда передача данных осуществляется с задержкой, стрелка рисуется с наклоном, а смещение по оси времени начала и конца стрелки, обозначающей передачу данных, равняется задержке передачи данных. Время работы ФУ выделяется жирной линией на оси абсцисс выделяется жирной линией.

***t***

***t***

***t***

***t***

***Регистры***

***Умн1***

***Умн2***

***Сум1***

**3**

**2**

**1**

**5**

**8**

**7**

**Ожидание устройством второго операнда**

**Вычислительная работа устройства**

**x B C A**

**4**

**Y**

**9**

**10**

Время передачи

данных между ФУ

Рис. 7. Диаграмма Ганта для описания параллельного вычислительного процесса

В данной системе время пересылки – 1 такт, поэтому-то линии, обозначающие передачу, здесь наклонные, что иллюстрирует задержку пересылки данных между ФУ. Время работы сумматора – 2 такта (единиц модельного времени), а умножителя – 9 тактов.

Существенное преимущество ВС с распределенным управлением – самораспараллеливание алгоритма и самовыбор одного из возможных альтернативных путей получения конечного результата. Например, вычислить площадь треугольника можно сразу несколькими способами: по двум сторонам и углу, по стороне и двум углам, по трем сторонам (формула Герона) и т.д. По какой ветке алгоритм найдет площадь треугольника первым, зависит от исходных данных и времени выполнения ветви алгоритма: если время вычислений по одной из ветвей меньше и для вычислений имеются все необходимые данные, то вычислительный процесс по ней быстрее дойдет до получения окончательного результата.

Важный параметром для оценки качества работы ВС является коэффициент параллелизма: т.е. сколько в среднем в единицу времени параллельно работает исполнительных устройств. Для вычисления коэффициента параллелизма (S) следует подсчитать количество параллельно работающих устройств в каждом такте и разделить полученное число на количество тактов работы алгоритма. Данный коэффициент показывает, сколько в среднем в единицу времени ФУ работают параллельно. Если S=1, то ВС работала в последовательном режиме; если S=2, то в среднем в ВС параллельно работало два ФУ и т.д. Для вычислительной системы, рассмотренной выше, S=32/24=4/3=1,33.

При недостаточном объеме вычислительных ресурсов в ОА ВС можно применить планирование вычислений. В этом случае вычислительное ядро ВС состоит из набора ИУ, а ФУ, ассоциированные с узлами потокового графа, представляют собой виртуальные устройства, которые могут находиться в активном или пассивном состоянии. Активны ФУ в том случае, когда им выделается ИУ (например, процессорное ядро). Активация ФУ происходит, когда на его порт поступает токен. Для управления распределением вычислительных ресурсов выделяется устройство коммутатор-планировщик. Он пересылает милликоманды между ФУ, и выделает вычислительные ресурсы для них. В том случае, если имеется свободное ИУ при коммутатор-планировщик выделяет ИУ для ФУ и передает ему милликоманду; после выполнения вычислений ФУ освобождает ИУ и переходит в неактивное состояние. Если же все доступные ИУ распределены между ИУ, то милликоманда помещается в очередь ожидания ресурсов. При освобождении ИУ планировщик-коммутатор отдает ядро тому ФУ, для которого предназначается милликоманда из вершины очереди ожидания ресурсов. Данная архитектура основывается не теории процессных сетей Кана (KPN) […]. Архитектура описанной выше ВС приведена на рис. 8.

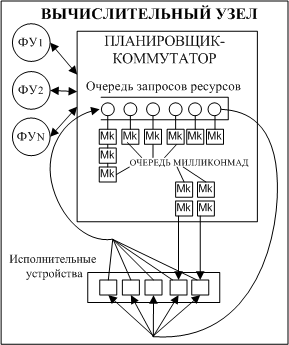


Рис. 8 Архитектура ОА ВС с распределением вычислительных ресурсов

Целью моделирования ОА ВС, которое будет осуществляться в ходе лабораторного практикума, является определение оптимального объема вычислительных ресурсов (количества ИУ) для решения определенной вычислительной задачи.

**2.2 ОА-язык программирования**

Для управления ОА ВС был разработан специализированный языка программирования – объектно-атрибутный язык (ОА-язык). Язык описывает вычислительный процесс в data-flow парадигме. Он позволяет управлять ФУ, входящие в состав ОА-ВС, а также описывать сложные информационные структуры, обработку которых осуществляет ОА-ВС. Язык может работать в режиме как компилятора, так и интерпретатора. При виртуальной реализации ОА-ВС (когда ФУ являются программами) язык позволяет динамически создавать виртуальные устройства и выгружать их при отсутствии необходимости их дальнейшего присутствия в составе ВС. Язык также применяется для описания архитектуры ВС (ее состав и информационные связи между устройствами). Две основные функции ОА-языка: настройка ФУ, входящих в состав ОА ВС, и передача исходных данных для ФУ. При виртуальной реализации язык имеет третью функцию: описание виртуальных ФУ, входящих в состав ОА ВС.

При осуществлении практической части работы программирование на ОА-языке будет осуществляться в среде ОА программирования и моделирования. Она позволяет описывать архитектуру виртуальной ОА ВС (создание и настройка ФУ), выдавать исходные данные для вычислений виртуальной ВС, собирать результаты работы ВС, давать команды управления ОА ВС непосредственно во время ее работы (режим интерпретатора).

Перечислим основные синтаксические конструкции ОА-языка

Целая константа. Записывается как последовательность цифр, перед число может присутствовать обозначение знака числа: «+» или «-».

Дробная константа – дробная часть отделяется от дробной с помощью знака «.».

Текстовая константа заключается в знаки кавычек.

Именованные константы. Задаются с помощью знака #. Именованные константы записываются с новой строки. Слева от знака «#» располагается имя константы, слева – значение. Значение может быть любого из следующих типов: логический, целое или дробное число, символ, строка. Именованная константа не может хранить ссылку на сложную информационную конструкцию в памяти, например, массив.

Переменная описывается с помощью знака «=»: справа мнемоника переменой, слева – значение для инициализации переменной. Тип переменной определяется константой для инициализации. Например, если константа целая, то и переменная будет иметь целый тип, если строка, то и переменная будет строкового типа и т.д.

Функциональное устройство (ФУ) – виртуальное устройство, обрабатывающее данные, приходящие в виде милликоманды. В ОА-языке каждое ФУ имеет собственное имя, идентифицирующее его. Имя присваивается ФУ при его создании. Создание осуществляется с помощью команды ОА-языка NewFU{Name=Имя ФУ FUType=название типа ФУ, Hint=описание ФУ}. Например, команда

NewFU={Mnemo="M1\_1" FUType=FUStreamFloatAlu Hint=” ФУ первой вычислительно нити ”} (1)

создает ФУ типа FUStreamFloatAlu с именем «M1\_1» и подсказкой «ФУ первой вычислительно нити». Подсказка всплывает во время наведения курсора мыши на название ФУ в консоли программы. Описание подсказки при создании ФУ необязательно.

Миллиикоманда (МК) – простейший токен, предназначенный для описания обмена данных между ФУ. МК состоит из двух полей: атрибут и нагрузка. Нагрузка представляет собой данные, представляющие собой операнд, а атрибут – индекс этих данных, по которым устройство-вычислитель идентифицирует эти данные и принимает решение по способу их дальнейшей обработки. МК обозначается с помощью знака «=», слева от которого указывается атрибут, а справа – нагрузка. Атрибут включает в себя два поля – имя ФУ и атрибут данных. Например, Dev.Mul=10, обозначает, что данная МК предназначается для ФУ с именем «Dev», имеет метку «умножаемое» (от агл. Multiplication – умножение), и передает для ФУ целую константу 10. МК является разновидностью информационной пары (ИП), предназначенной для управления ФУ. Другая разновидность ИП – информационная ИП.

Информационная ИП предназначена для описания токена, где атрибут идентифицирует данные (такая ИП не обрабатывается ФУ непосредственно). Также имеет два поля: атрибут и данные. Однако атрибут для данной разновидности ИП представляет собой мнемонику без указания ФУ. Например, Atr=10, где Atr – мнемоника атрибута. В ОА ВС каждый атрибут представляет собой уникальный индекс, однако для удобства программирования он в ОА-языке заменяется мнемоникой.

Атрибут – мнемоника атрибута для информационной ИП. Некоторые атрибуты уже прописаны в ОА-среде (объявленные атрибуты можно увидеть на вкладке «Атрибуты» левой панели ОА-среды. Однако имеется возможность объявления пользовательских атрибутов. Объявления могут быть двух типов: с автоматической индексацией и с принудительной индексацией. В первом случае ОА-среда сама присваивает индекс атрибуту, во втором пользователь назначает индекс атрибут сам. В первом случае необходимо записать мнемонику атрибута, во втором после мнемоники записывается знак “\*” и после него указывается индекс, например, Art\*-120. *Примечание: желательно, чтобы индексы атрибутов были отрицательными, т.к. ОА-среда воспринимает положительные индексы атрибутов, как атрибуты МК, а отрицательные как атрибуты информационных ИП.*

Именованная нагрузка. Нагрузка ИП по сути представляет собой переменную, хранящую данные или указатель. Чтобы программист мог получить доступ к ней, имеется возможность присвоить нагрузке имя и затем получить доступ к ней как к переменной. Для этого после знака «=» указывается мнемоника нагрузки, а в скобках записывается значение для инициализации нагрузки. Например, «Cons.Out=temp(0)».

Информационная капсула (ИК) – множество (последовательность) МК или атрибутированных данных. Обозначается с помощью фигурных скобок «{}», между которыми записывается последовательность МК или атрибутированных данных. Например, в команде создания ФУ (1) с помощью атрибутированных данных (атрибут ассоциируется с данными с помощью знака «=») передаются параметры создаваемого ФУ. *Примечание: знак «=» за пределами ИК (т.е. вне фигурных скобок) воспринимается компилятором как знак присвоения или инициализации переменных, в пределах ИК он интерпретируется как разделитель атрибута и нагрузки МК.* ИК могут выступать в качестве нагрузки ИП, в этом случае в описание ИК добавляется после знака «=». Например, ALU.Prog={ALU.OutMk=ALU.Mul ALU.OutMk=Cons.LnOut} (2).

ИК могут быть именованными или неименованными. Имя ИК в указывается перед первым знаком «{» при описании ИК. Например, (1) можно изменить таким образом, чтобы МК была именованной:

ALU.Prog=Prog{ALU.OutMk=ALU.Mul ALU.OutMk=Cons.LnOut},

где Prog – имя ИК. Именованные ИК также могут быть указаны отдельно без нагрузки. Для этого необходимо записать отдельную ИК с именем, такая ИК не может входить в описание другого ИК. Имя ИК может быть записано в качестве нагрузки ИП, например, Cons.OutLn=Prog. Prog – это указатель на ИК.

Удаление любой мнемоники (указателя на ИК, переменной или именованной константы) осуществляется с помощью знака «!», стоящего после мнемоники. Мнемоника может указываться отдельно («temp!»), так и в качестве нагрузки ИП («Cons.out=temp!»).

Поздняя инициализация нагрузки ИП. Иногда при написании программ необходимо указать адрес переменной или нагрузки, куда ФУ будет выдавать результат своей работы; однако именование переменной или нагрузки происходит в ниже в программе. Для этого в можно указать мнемонику переменной или нагрузки до его объявления. Например, «ALU.Out=temp Cons.OutLn={Atr=temp(0)!}». В данном случае АЛУ выдает результат своей работу, которой записывается в нагрузку ИП с атрибутом «Atr», а затем ФУ «Консоль» осуществляет вывод данной ИП на экран. Использование поздно инициализированной переменной может быть осуществлено несколько раз.

ОА-граф. Несколько ИК могут быть объединены в одну граф (сеть) посредством указания ссылок на ИК в нагрузках ИП, входящих в состав ИК. Для описания графа тип дерево достаточно описывать ИК более низкого уровня в качестве нагрузки ИП. Например,

Tree{

Atr={Atr=1 Atr={ Atr=2 Atr=3} Atr=4}

Atr={Atr=5 Atr={Atr=6}}

}

На рис. … представлена структура описанного графа-дерева.



Рис. … Пример ОА-дерева

Список используется для быстрого описания ОА-графа типа дерево. Для обозначения ИК одного уровня применяется знак «>». Список может быть как именованным, так и нет; также список может быть описан в качестве нагрузки. Например:

// Свободный список

List

>{Atr=11, Atr=12}

>ListNmemo{Atr=21, Atr=22 Atr=23}

>{Atr=31, Atr=31}

Cons.Out=List

// Список в качестве нагрузки

Cons.Out=

>{Atr=11, Atr=12}

>ListNmemo{Atr=21, Atr=22 Atr=23}

> ListNmemo

На рис. … иллюстрируется структура списка, List.



Рис. … Структура двух перекрестных ОА-списков

Для создания ОА-графа произвольной структуры необходимо воспользоваться именованием ИК, и затем указать это имя в нагрузке ИП. Например,

Graph{

Atr=Label{Atr=1 Atr={ Atr=Label2 Atr=3 } Atr=4}

Atr= Label2{Atr=5 Atr={Atr=Label}}

}

Структура описанного графа приведена на рис. … .



Рис. … Структура ОА-графа с произвольной топологией

ОА-списки и ОА-графы могут найти применение для различных целей. Например, в системах искусственного интеллекта, а также в системах компиляции и интерпретации языков программирования.

Области применения ОА ВС в сочетании с ОА-языком достаточно обширные: высокопараллельные вычисления, распределенные вычислительные системы, распределенные системы автоматического управления, системы анализа естественного языка (NLP), системы искусственного интеллекта и т.д.

Описание потокового графа алгоритма производится с помощью программирования каждого ФУ, следующим образом

1. если в узел входит дуга из вершины, обозначающей константу, то необходимо описать милликоманду вида: Мнемоника\_ФУ. Милликоманда =Мнемоника\_константы.

2. если из узла потокового графа выходит дуга, то в ОА-программе её необходимо описать следующим образом:

Мнемоника\_ФУ.ReseiverMkSet=Мнемоника\_ФУ-приемника.Милликоманда

Следующим этапом является непосредственно описание алгоритма решения вычислительной задачи. Описание алгоритма работы каждого ФУ осуществляется следующим образом:

1. если в узел входит дуга из вершины, обозначающей константу, то необходимо описать милликоманду вида: Мнемоника\_ФУ. Милликоманда =Мнемоника\_константы.

2. если из узла потокового графа выходит дуга, то в ОА-программе её необходимо описать следующим образом:

Мнемоника\_ФУ.ReseiverMkSet=Мнемоника\_ФУ-приемника.Милликоманда



Рис. … Фрагмент потокового графа

Например, для фрагменту потокового графа, представленного на рис. …, где каждому каждое ФУ именовано, чтобы к нему можно было обращаться через ОА-язык. Следует описать задать такую ОА-программу:

1: Mul1.Mul1=a

2: Mul1.Mul2=a

3: Mul1.ReceiverMkSet=Mul1.1\_Mul1

4: Mul1.1\_Mul2=a

5: Mul1.1\_ReseiverMkSet=Sum1.Sum2

6: Sum1.Sum1=a

7: Sum1.ReceiverMkSet=Console.LnOut

В строках 1,2,4 и 6 описывается поступление в качестве входных данных, констант. В строчке 7 описывается вывод результата вычислений на консоль (милликоманда Console.LnOut). Рекомендуется вынести все милликоманды, описывающие поступление констант (исходных данных) в конец ОА-программ, что в дальнейшем обеспечит удобство доработки ОА-программы для проведения имитационного моделирования. В том случае, когда на одном ФУ реализуются несколько операций, то для каждой операции перед мнемоникой милликоманды, ставится номер команд с последующим знаком «\_»; для первой команды номер не ставит (подразумевается «0\_»). В строках 1,2,3 – ведется настройка первой команды ФУ Mul1, в строках 4,5 – второй команды.

Ниже приведен формализованный синтаксис ОА-языка в расширенной форма Бекуса-Наура [Пилецкий И.И. Методы трансляции: метод. пособие для студ. спец. 1-13 03 04 «Информатика» всех форм обуч. / И.И. Пилецкий, В.В. Шиманский. – Минск : БГУИР, 2012. – 88 с.]:

Sumb = ”a”| “b” | … | “z” | “A” | “B” | … | “Z”

Fig = “0” | “1” | “2” | “3” | “4” | “5” | “6”| “7” | “8” | “9”

Мнемо = Sumb ( Sumb | Fig )

Digit = [Fig {Fir}] [“.”Fig {Fir}]

Const = Digit | “ ” ” [{Sumb | Fig}] “ “ ”

Mk = Mnemo “.” Mnemo

Atr = Mk | Mnemo

IP = Atr “=” Load

Load = (Const | [Mnemo] IC)

IC = “{” {IP} “}”

NewFU= “NewFU={Name=” Name “ “FUType=” Const “}”

Prog = { NewFU | (Mk “=” Load ) | (Mnemo “=” Const) | (Mnemo “#” Const)}

**2.3 Организация параллельных вычислений в ОА вычислительной системе с ограниченными вычислительными ресурсами**

Негативным свойством data-flow ВС является потребление больших вычислительных ресурсов: в идеале для обеспечения максимальной производительности ВС каждому ФУ (узлу потового графа) должно соответствовать свое ИУ. Однако современная технология производства вычислительной аппаратуры не в состоянии обеспечить ВС такими колоссальными ресурсами (в настоящее время на процессорном кристалле находятся до нескольких десятков ядер). Поэтому в data-flow вычислительной системе встает необходимость распределения вычислительных ресурсов. В ОА архитектуре эта проблема решена с помощью применения специализированного устройства «Планировщик-коммутатор» (или сокращенно «Шина» («Bus»)). Оно является коммутатором всех МК, которыми обмениваются между собой ФУ, входящие в состав ОА ВС. При передаче МК ФУ пересылает ее на Шину. Шина по атрибуту МК производит определение ФУ-адресата и пересылает ему ее. При таком решении атрибут МК делится на два поля: номер ФУ-адресата и атрибут данных. По первому полю Шина и определяет адресата. Шина может быть реализована как программным, так и аппаратным способом.

При ограничении количества ИУ (процессорных ядер), Шина производит распределение ИУ между ФУ. ***Внимание***: под ФУ мы подразумеваем виртуальное вычислительное устройство, ассоциированное с узлом потокового графа; под ИУ – аппаратное устройство, производящее вычисление. ФУ может находиться в двух состояниях: активное и пассивное. В пассивном состоянии ФУ не имеет в распоряжении ИУ и его контекст (совокупность внутренних регистров) хранится в оперативной памяти. Переход ФУ в активное состояние происходит, когда ему предоставляется вычислительный ресурс, а необходимость в вычислительном ресурсе возникает в момент прихода в ФУ МК, которую необходимо обработать. Все МК проходят через шины, поэтому она и занимается распределением ИУ. В распоряжении Шины имеется пул ИУ мощностью N, т.к. в вычислительном узел находится N устройств-вычислителей. Пусть все ИУ будут универсальными. Тогда, если в момент прихода МК в пуле имеются свободные ИУ, то Шина представляет один из них в качестве ресурса для ФУ-адресата МК. Если же свободных ИУ нет, то МК становится в очередь ожидания освобождения ресурсов. Если ФУ-адресату уже выделен ИУ и оно занято обработкой предыдущей МК, то новая МК также становится в очередь ожидания ресурсов (Рис. …).

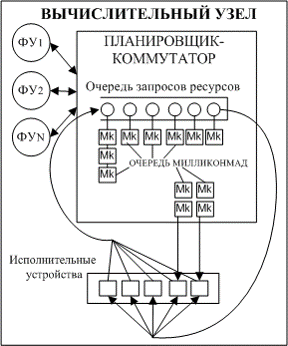


Рис. … Планировщик-коммутатор в состав ОА вычислительной системы

Выше описанная организация распределения вычислительных ресурсов имеет один недостаток. Если потом данных будет слишком большой, то в ВС может не хватить оперативной памяти, чтобы разместить в ней все МК, адресованные к пассивным ФУ.

Первый лабораторный практикум будет посвящен моделированию параллельного data-flow вычислительного процесса, выполняемого на базе ВС с ограниченными вычислительными ресурсами. Архитектура ВС, моделирование которой будет в ней производиться, идентична архитектуре, описанной выше.

**2.4 Трансляция языков на базе ОА вычислительной системы**

ВС ОА-архитектуры также подходит и для создания систем трансляции языков программирования высокого уровня. ОА-транслятор позволяет описать все этапы трансляции: лексический и синтаксический анализы, построение внутреннего представления данных, семантический анализ, оптимизацию кода и трансляцию на другой язык (в частности, машинный код). Данной тематике посвящен лабораторный практикум 2.

Система трансляции языка строится на базе распознающего автомата [Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман Компиляторы. Принципы, технологии и инструментарий. - 2 изд. - М.: Вильямс, 2016. 1178 c]. Автомат имеет множество состояний и может переходить из одного состояния в другой под воздействием текущего символа, расположенного на входной ленте автомата (на этой ленте располагаются символы транслируемого языка). Текущим считается символ под указателем автомата. В начальный момент указатель стоит на первом символе последовательности, затем через каждый такт работы автомата передвигается на один символ вправо и так до конца ленты. Если автомат оказывается в одном из конечных состояний, то последовательность символов считается синтаксические правильной (принадлежащей языку). Для синтаксического анализа и трансляции языков, относящихся к классу контекстно-свободных (в этом классе находятся практически все языки высокого уровня) необходимо автомат с магазинной памятью (МП-автомат). Магазин представляет собой стек, где могут храниться служебные символы, используемые в основном для обозначения вложенных скобочных конструкций языка.

Формально автомат-транслятор с магазинной памятью представляет собой семерку:

M = (Σ, Q, Z, Y, q0, z0, F), где

* Σ - конечный алфавит входных символов;
* Q - конечное множество состояний автомата;
* Z - алфавит памяти (магазина);
* Y http://citforum.ru/programming/theory/serebryakov/cmsy10-12.gif Q - множество конечных состояний автомата;
* q0 http://citforum.ru/programming/theory/serebryakov/cmsy10-32.gif Q - начальное состояние автомата;
* z0 http://citforum.ru/programming/theory/serebryakov/cmsy10-32.gif Z – специальный символ дна магазина (маркер);
* F – функция переходов, задающая отображение множества Σ×X×Z×Z×Q.

Схема автомата-транслятора приведена на рис. … .

Изображение выглядит как снимок экрана

Автоматически созданное описание

Рис. … Автомат-транслятор с магазинной памятью

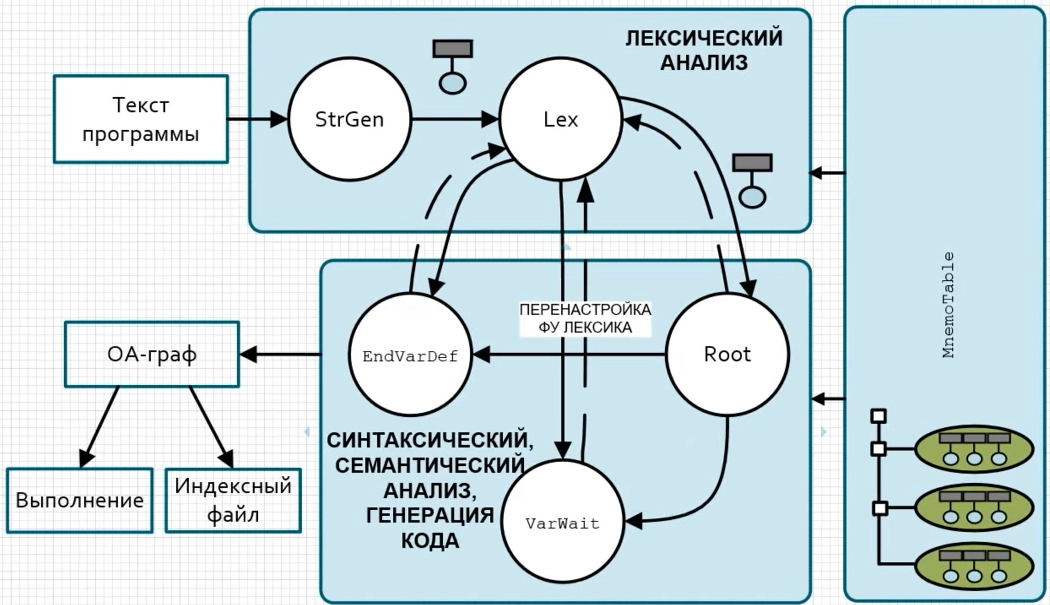


Рис. … Организация ОА-транслятора языка

В ОА ВС распознающий автомат представляет собой совокупность ФУ, выполняющих функцию автомата-транслятора (ОА-автомат) [Салибекян С.М. Реализация автоматной парадигмы вычислений на объектно-атрибутном базисе // Прикладная информатика. 2017. Т. 12. № 2 (68). С. 103-115]. Перечислим типы ФУ и их функции в составе ОА-автомата (рис. …) [Салибекян С.М., Панфилов П.Б. Анализ языка с помощью объектно-атрибутного подхода к организации вычислений // Программная инженерия. №1, 2013 – стр. 9-16.].

**Генеретор строк (StrGen)** предназначен для чтения входной последовательности символов (текста транслируемой программы) их любого источника: файл, консоль, сеть и т.д. StrGen считывает очередную строку текста и передает ее для дальнейшего разбора на лексический анализатор.

**Лексический анализатор (Lex)** производить лексический анализ текста, заключающийся в выделении лексем: мнемоник, знаков-разделителей, чисел и т.д. Lex также определяет тип лексемы и формирует токен (ИП), где атрибутом является тип лексемы, а нагрузкой – сама лексема. Токен затем передается на ФУ-состояние автомата. Адрес текущего ФУ-состояния хранится в специализированном регистре, входящим в контекст ФУ Lex. Для того, чтобы автомат сменил текущее состояние, необходимо записать другой адрес в данный регистр.

**ФУ-состояние**. Данное ФУ выполняет роль состояния автомата. Оно имеет тип FUList – список ИК. Каждая линия списка задает переход из данного состояния состояние другое. В начале ИК находятся ИП с описанием лексем, по которым происходит переход, а следом за ними располагаются МК переходной программы. Эта программа запускается в том случае, когда описание лексемы, поступившей от Lex, совпадает с одной из лексем в ИК. Переходная программа реализует определенный алгоритм, соответствующий данному этапу анализа языка программирования: синтез внутреннего представления данных компилятора, выдача выходного текста, оптимизация и т.д. Также в переходную программу входит МК записывающая в Lex адрес нового ФУ-состояния, куда Lex должен передавать следующую, выделенную из текста, лексему.

**Стек автомата** реализуется также на базе FUList. Каждая линия этого ФУ содержит описание объекта, который автомату необходимо запомнить для осуществления дальнейшего анализа.

**Таблица описания переменных и других объектов**. В транслируемой программе могут встречаться объявления переменных. После объявления описание переменной (мнемоника, тип данных, ссылка на ячейку оперативной памяти, где хранится переменная) помещаются в данную таблицу. Если в процессе трансляции в тексте повторно встретится мнемоника данной переменной, то транслятор найдет ее в данной таблице и будет интерпретировать соответственно ее описанию. Таблица переменных также реализуется на базе FUList.

Остановимся более подробно на описании FUList. Он содержит в своем контексте ссылку на ОА-список, может выполнять некоторые манипуляции над ним: удалять и добавлять ИК в список, осуществлять поиск информации, а также запуск программ, записанных в ИК списка. Например, для поиска данных применяется МК FindAnd (найти по правилу «И»). В нагрузку этой МК помещается ИП, которую необходимо найти в одной из ИК списка. ИК списка просматриваются последовательно от верхней к нижней. Если в ИК будет обнаружена ИП, что было передана в нагрузке FindAnd, то дальнейший поиск прекращается. Если в данной ИК имеется последовательность МК, то FUList запускает их на выполнение.

Пример реализации ОА системы семантического анализа приведен в разделе 4. Сейчас же приведем описание проектирования подобной системы в классической парадигме вычислительного процесса и использованием switch-технологии.

Для синтаксического анализа можно применять и классическую архитектуру ВС, применив так называемую switch-технологию автоматного программирования [Поликарпова Н.И., Шалыто А.А. Автоматное программирование. – СПб: Питер, 2011. – 176 с.]. Switch-технология предполагает следующую программную реализацию автоматной парадигмы. Каждому состоянию автомата присваивается номер. В программу вводится переменная, хранящая номер состояния. Пусть эта переменная будет именоваться S. Далее в программе организуется цикл считывания данных, внутри которого находится оператор множественного ветвления switch по переменной S. Каждая альтернатива выбора соответствует состоянию автомата-транслятора, где описывается обработка входных данных, генерация выходной последовательности символов, и переход в другое состояние автомата. Переход заключается в присвоении переменной S номера нового состояния. Далее цикл считывания нового символа или лексемы повторяется, снова задействуется оператор switch и т.д.

Для примера приведем программу проверки правильности следующей синтаксической конструкции, записанной в расширенной форме Бекуса-Наура (РБНФ) […]:

I “=” ( ( (I | DF) (“+” | “-” | “/” | “\*”) (I | DF) ) | (I (“and” | “or” | xor”) I) )

I “=” ( ( (I | DF) (“+” | “-” | “/” | “\*”) (I | DF) ) | (I (“and” | “or” | “xor”) I) )

…

Где

Alpha= “a”| “b” | “c” | … | “z” | “A” | “B” | … | “Z”

Figure= “0”| “1”| “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

Id – имя переменной; I=Alpha (Alpha | Figure)\*;

DF – дробное десятичное число DF= Figure (Figure)\* “.” Figure (Figure)\*.

Составим граф состояний для распознающего автомата (автомата, проверяющего синтаксическую правильность языка) синтаксического анализатора. Пусть в программе лексический анализ обеспечивает функция «Lex», которая при вызове выдает очередную лексему. Пронумеруем все вершины автомата. Граф приведен на рис. … . Дуги графа помечены лексемами, по которым автомат производит переход от одного состояния к другому. Состояние с индексом 0 является начальным и конечным. Далее приведена программа, реализованная по Switch-технологии.



Рис. … Граф состояний распознающего автомата

#include <iostream>

include <string>

using namespace std;

string Lex() // Подпрограмма лексического анализа

// Возвращается тип лексемы в строковом виде

{

//…

}

int main()

{

int S=0; // Инициализация для перевода автомата в начальное состояние

while(true)

{

if(S==-1) break; // Выход по состоянию ошибки

stirng L=Lex();

if(L=="@") // Проверка конечного символа строки

break;

switch(S)

{

case 0:

if(L=="Id") // Лексемы идентификатор?

{

S=1; // Переход в состояние с индексом 1

// ... Возможное описание действий компилятора

//в данном состоянии автомата

}

else

{

cout<<"Identifier is not found\n";

S=-1; // Состояние ошибки

}

break;

case 1:

if(L=="=") // Лексемы идентификатор?

S=2; // Переход в состояние с индексом 2

else

{

cout<<"Equal sign is not found\n";

S=-1; // Состояние ошибки

}

break;

case 2:

if(L=="Id") // Лексемы идентификатор?

S=3; // Переход в состояние с индексом 3

else if(L=="DF")

S=4; // Переход в состояние с индексом 4

else

{

cout<<"Operand is not found\n";

S=-1; // Состояние ошибки

}

break;

case 3:

if(L=="and" || L=="or" || L=="xor") // Лексемы идентификатор?

S=5; // Переход в состояние с индексом 5

else if(L=="+" || L=="-" || L=="\*" || L=="/")

S=6; // Переход в состояние с индексом 6

else

{

cout<<"Operation is not found\n";

S=-1; // Состояние ошибки

}

break;

case 4:

if(L=="+" || L=="-" || L=="\*" || L=="/") // Лексемы идентификатор?

S=6; // Переход в состояние с индексом 6

else

{

cout<<"Operation is not found\n";

S=-1; // Состояние ошибки

}

break;

case 5:

if(L=="Id") // Лексемы идентификатор?

S=0; // Переход в состояние с индексом 0

else

{

cout<<"Operation is not found\n";

S=-1; // Состояние ошибки

}

break;

case 6:

if(L=="Id" || L=="DF") // Лексемы идентификатор?

S=0; // Переход в состояние с индексом 0

else

{

cout<<"Operation is not found\n";

S=-1; // Состояние ошибки

}

break;

}

if(S==0)

cout<<"Correct\n";

else

cout<<"No correct\n"

}

return 0;

}

Switch-технология позволяет реализовать любой распознающий автомата или автомат-транслятор. Однако у данной технологии есть определенные недостатки. Во-первых, индексация состояний не очень удобна для программиста, т.к. человек лучше воспринимает информацию в виде текста. Однако данный недостаток можно преодолеть с помощью именованных констант, или с помощью enumeration. Во-вторых, достаточно громоздким выглядит описание переходов из одного состояния в другое. Лексемы после этапа лексического анализа, как правило, представляются в виде токена, где атрибут которого хранит тип лексемы, а в нагрузке находится указатель на содержание лексемы. В результате, классическая программа значительно усложняется по сравнению приведенной выше программой. В-третьих, программу нельзя разделить на несколько модулей, что при наличии большого количества

ОА-подход позволяет значительно упростить программу и сделать ее более эргономичной для программиста, т.к. ОА-архитектура приспособлена для работы с токенами (ИП). Описание лексемы на фазе синтаксического анализа представляется в виде ИП. Возможные переходы из состояния в состояние автомата-транслятора можно описать в виде ОА-списка. Содержание таблицы описаний лексем также представляется в виде ОА-списка. Состояния автомата ассоциируются с ФУ-состояниями, а в ОА-языке каждое ФУ имеет свое имя, т.е. состояния автомата именуются (более привычно для человека, чем индексы). Описание автомата можно разделить на несколько программных модулей, что нельзя сделать при switch-технологии). Внутреннее представление данных для компилятора/интерпретатора представляется в виде ОА-графа. Таким образом, программа и представление данных представляются единообразным способом, ОА-программа становится более эргономичной (см. раздел 4).

**2.5. Другие вычислительные задачи, решаемые на базе ОА архитектуры**

В настоящем учебном пособии описаны два лабораторных практикума, касающиеся организации параллельных вычислений и систем трансляции языков программирования высокого уровня. Однако ОА архитектура имеет еще достаточно много приложений. Перечислим их.

**Распределенные вычислительные системы**

Преимуществом Data-flow ВС является то, что операнды, участвующие в вычислениях, не идентифицируются, как в классических ВС, с помощью адреса ячейки оперативной памяти, где они хранятся; а идентифицируются служебной информацией в токене. В ОА ВС в качестве служебной информации применяется атрибут ИП. Поэтому вычислительный процесс, запускаемый на распределенной ВС (ВС, состоящая из вычислительных узлов с локальной памятью), реализуется «естественно», т.е. ВС «не замечает», что она распределенная и работает как единое целое. В классической же ВС распределенный вычислительный процесс организуется достаточно сложно. Например, можно использовать библиотеку MPI (Message Passing Interface) [Модели параллельного программирования. — М.: СОЛОН-ПРЕСС, 2012. — 384 с.], с очень громоздким интерфейсом; также применяется механизм RPC (Remote Program Call) [Стин ван М., Таненбаум Э. С. Распределенные системы / пер. с англ. В. А. Яроцкого. – М.: ДМК Пресс, 2021. – 584 с.]; библиотеки CORBA и DCOM [Роберт Орфали, Дан Харки, Джери Эдвардс Основы CORBA: Пер. c англ. – М.: МАЛИП, Горячая Линия - Телеком, 1999 – 316 с.]. Все вышеперечисленные инструменты весьма усложняют процесса программирования распределенных ВС и не обеспечивают масштабируемость программы (масштабируемость – это свойство программы с малыми доработками, или вообще без них быть перенесенной на другую распределенную ВС с другим количество входящих в нее вычислительных узлов и, возможно, другой топологией).

Например, в data-flow ВС ОА архитектуры распределенные вычисления могут быть организованы следующим образом [S.M. Salibekyan, P.B. Panfilow Object-attribute architecture for design and modeling of distribute automation system. // Automation and remote control. Volume 73, 2012, Number 3, 587-595, DOI: 10.1134/S0005117912030174] [Салибекян С. М., Панфилов П. Б. Вопросы автоматно-сетевого моделирования вычислительных систем с управлением потоком данных // Информационные технологии и вычислительные системы. 2015. № 1. С. 3-9.] (рис. …). Разделим ВС на две части: ОА-образ программы и ОА-платформа. Первая часть – это совокупность ФУ (т.е. виртуальных устройств), их информационные связи, поведение (загруженная программа в ФУ) и сложно структурированная информация для обработки. Вторая – это аппаратная часть ВС: вычислительные узлы с локальной оперативной памятью, служебные программы, обеспечивающие функционирование вычислительного узла (системные программы), и коммуникационное оборудование, эти узлы связывающее. На каждом вычислительном узле расположен ФУ «Манршрутизатор» («Router»); маршрутизатор относится к ОА-платформе, т.к. он привязан к вычислительному узлу. Он обеспечивает маршрутизацию МК, передаваемых по распределенной ВС.



Рис. … Распределенная ОА ВС

В ОА-образ входят ФУ типа Шина. В задачу Шины входит коммутация МК между ФУ, подсоединенных к ней. Шину с подключенным к ней ФУ будем называть сегментом. Но если ФУ оказывается вне этого сегмента (в том числе расположенном на другом вычислительном узле), то Шина перенаправляет МК Маршрутизатору, который и перенаправляет МК по нужному маршруту. Локация МК определяется по атрибуту МК: каждому сегменту выделяется диапазон индексов, поэтому по МК можно узнать номер сегмента, куда «направляется» МК. В состав Маршрутизатора входит таблица маршрутизации, где находятся сведения, по какому маршруту следует направлять МК. Шлюз – это коммуникационное оборудование, непосредственно осуществляющее передачу данных по каналу связи.

ОА-образ можно перенести на ОА-платформу с любым составом вычислительных узлов и любой топологии. На рис. … а) б) представлены два варианта переноса ОА-образа на две ОА-платформы. В разных вариантах разные сегменты поселяются на разные вычислительные узлы. Напротив каждого канала передачи данных, управляемых роутером, приведен фрагмент его таблицы маршрутизации. В нем записаны номера сегментов, МК для которых необходимо отправлять в данный канал.

а) б)

Рис. … Перенос ОА-образа на различные ОА-платформы

Такая организация распределенной ВС обеспечивает масшрабируемость, т.к. сегменты ОА ВС могут быть легко «переселены» на другие вычислительные узлы: при этом необходимо только перенастроить таблицы маршрутизаторов. Также легко и осуществить запуск ОА образа на ВС другой конфигурации (иное количество вычислительных узлов, иная топология сетевых соединений).

Распределенные ВС применяются в нескольких областях. Во-первых, высокопроизводительные вычисления (Height Performance Computing (HPC)). В этой области необходимы большие вычислительные мощности, которые не может обеспечить централизованная ВС. Во-вторых, это – распределенные системы автоматизации: авиационные, автомобильная, судовая электроника; автоматизированные системы управления производством (АСУ ТП), системы управления инженерными сооружениями, SCADA-системы. Пример системы управления на база ОА-подхода можно посмотреть в [Салибекян С.М., Шибаев Р.В. Управляющий автомат и dataflow парадигмы вычислительного процесса - необычный симбиоз // Системный администратор. 2017. № 6 (175). С. 84-87.].

При виртуальной реализации ОА-платформы, т.е. когда ФУ являются виртуальными и запускаются на базе классической ВС, вышеописанный подход позволяет создавать виртуальную машины (наподобие JAVA), ведь виртуальное ФУ является прослойкой между ОА-программой и аппаратной частью ВС. Таким образом, вычислительные узлы распределенной ВС могут быть гетерогенными (разной архитектуры и производительности).

**Семантическая сеть**

ОА-граф может быть использован для реализации семантического сети [Джозеф Джарратано, Гари Райли, Экспертные системы. Принципы разработки и программирование // 4-е издание / Вильямс, 2007. – 1152 с.]. Семантическая (смысловая) сеть представляет собой ориентированный граф, применяемый для описания сложно структурированного объекта или системы. Каждая вершина сети ассоциируется с каким-либо подобъектом системы, а дуги графа описывают смысловые связи между подобъектами. В случае применения в качестве семантической сети, описание подобъекта помещается в ИК, которая является узлом ОА-графа (в нашем случае семантической сети). Каждая ИП в ИК описывает одно из свойств объекта: атрибут идентифицирует свойство, а в нагрузке хранится значение свойства. Дуги семантической сети описываются с помощью ИП с указателем на другую ИК, а атрибут данной ИП задает метку (тип) семантической связи. Пример ОА семантической сети можно видеть на рис… .



Рис. … Пример ОА-семантической сети

В данная семантическая сеть применяется для описания семантической связи между объектами и их свойств. Атрибут Class служит для идентификации класса объекта. Location – локация (местоположение объекта), Location Type – соответственно, тип локации. Obj – объект отношений (т.е. активный объект), Subj – субъект (объект, на который направлено действие). Atr – атрибут (свойство объекта) , SubAtr – указатель на объект, для которого описано свойство. Способ описания пространственно-временных отношений объектов с помощью ОА семантической сети приведен в [Salibekyan, S, Panfilov, P. Database architecture for specifying and modeling spatio-Temporal relations // Annals of DAAAM and Proceedings of the International DAAAM Symposium Volume 2015-January, 2015, Pages 589-598 , Салибекян С.М., Петрова С.Б. Объектно-атрибутная модель представления пространственно-временных отношений между объектами // Прикладная информатика. 2016. Т. 11. № 3 (63). С. 103-115.].

С помощью такой семантической сети можно описывать объекты и системы любой структуры и состава. Сеть может быть применена системах искусственного интеллекта: компьютерное зрение, семантический анализ естественного языка, сетевые (графовые) базы данных, базы знаний и т.д.

**Граф-трансформирующая система**

ОА-архитектура позволяет создавать граф-трансформирующие системы (ГТС) […]. ГТС – это система, осуществляющая преобразование структуры графа. Если граф представляет собой семантическую сеть, то преобразуются как ее структура, так и информационное содержание. Подклассом ГТС являются графовые грамматики [König, B., Nolte, D., Padberg, J., Rensink, A. A tutorial on graph transformation // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics10800 LNCS, p. 83-104]. Графовая грамматика представляет собой набор правил преобразования графа (продукций). Продукция состоит из двух частей: правой и левой. Левая часть предназначена для описания графа шаблона, который ищется в host-графе (графе, подвергающимся преобразованию). Если такой подграф был найден (он называется материнским подграфом). Если во время работы ГТС будет найдет материнский подграф, то к нему будет применено преобразование, заданное правой частью продукции (рис. …). В качестве примеров можно привести следующие формализмы графовых грамматик: [Müller S. “Unifying everything: Some remarks on simpler syntax, construction grammar, minimalism and HPSG.” Language 89(4), 2013. pp.920–950], LinkGrammar [Link grammar. — 2013. — feb. — URL: <http://www.abisource.com/projects/link-grammar/>], атрибутная транслирующая грамматика Н. Хомского (АТ-грамматика) [Aho, Alfred Vaino; Lam, Monica Sin-Ling; Sethi, Ravi; Ullman, Jeffrey David (2006). Compilers: Principles, Techniques, and Tools (2 ed.). Boston, Massachusetts, USA: Addison-Wesley. ISBN 0-321-48681-1. OCLC 70775643].



Рис. … . Граф-преобразующая система

Формально ГТС можно представить следующим образом:

**GT={M, D, E}**, где

M и D – «материнский» («mother») подграф и «дочерний» («daughter») подграф (граф-шаблон),

E – механизм встраивания дочернего подграфа в трансформируемый граф H («host»).

Графовая грамматика разработана и для ОА-подхода, т.к. в некоторых задачах необходимо не только представлять сложно структурированные данные в виде ОА-графа, но его преобразование.

ОА-грамматика представляет собой четверку [Salibekyan, S. and Panfilov, P. Linguistic Processor Based on Object-Attribute Grammar. In Supplementary Proc.: The 6th International Conference on Analysis of Images, Social Networks, and Texts (AIST-2017), Moscow, Russia, July 27-29, 2017, CEUR Workshop Proceedings, ISSN 1613-0073, Vol-1975, pp.134-145, URL: <http://ceur-ws.org/Vol-1975/paper15.pdf>]:

OAG={A,L,P,G},

где A – алфавит атрибутов;

L – алфавит нагрузок ИП (в алфавит входят не только числа и строки, но и ссылки на ИК;

G – ОА-граф (список описаний лексем исходного языка);

P – правила преобразования ОА-графа.

Правила преобразования ОА-графа по аналогии с правилами формальной грамматики состоят из двух частей (левая и правая) и разделены знаком «->». В левой части выражения записывается линейный фрагмент списка исходных лексем, а в правой – фрагмент списка, на который необходимо заменить левый фрагмент. Для записи фрагментов ОА-списка исходных лексемы используются следующие обозначения:

a = l – обозначение ИП (a∍A, l∍L, где A – множество атрибутов, L – множество нагрузок ИП);

IpName{…} – ИК (IpName – имя ИК, которое можно использовать в качестве обозначения ссылки на ИК; между знаками «{» и «}» записываются ИП, входящие в ИК); имя ИК может и не указываться; ИК представляет собой цепочку ИП;

a=IpName – ИП с указателем на ИК в нагрузке (a ∍ A; IpName – имя ИК);

{…a1= IpName{a2, l2} …} – ИК, в которую входит ИП с указателем на другую ИК в нагрузке; имя ИК может и не указываться;

\* – конкатенация ИК (запись Ic1\*Ic2 обозначение показывает, что капсула Ic2 должна быть сцеплена с капсулой Ic1);

| – вариант (перечисление смысловых альтернатив);

( ) – группировка правил вариантов (знак «|» имеет более низкий приоритет, чем «\*»), т.е. правила, заключенные в круглые скобки, имеют бóльший приоритет, например, выражение IC (IC2 | IC3) эквивалентно IC\*IC2 | IC\*IC3;

[условие] – логическое условие, накладываемое на нагрузку ИП, например, Atr=[>0] – т.е. нагрузка должна быть больше нуля;

:: - операция перемещения ИК (т.е. описание ИК копируется на новую позицию в графа, а затем удаляется из host-графа).

В качестве примера работы ОА ГТС приведем пример ГТС для преобразования фрагмента семантического графа, задающего структуру семантической сети описания семантико-синтаксической структуры предложения, а точнее склейки словосочетания с наречием степени (adverb degree) и прилагательным (adjest). Например, «очень большой». На рис. … приведены фрагменты графа до а) и после б) преобразования. Как видно, семантические свойства (SemProp – semantic property) наречия степени (атрибут POS – это обозначение части речи (Part of Speach)) с атрибутом («степень» - degree) перемещаются в описание семантических свойств прилагательного.



Рис. … . Результат работы граф-трансформирующей системы

Данному преобразованию соответствует следующая запись ОА-грамматики:

{Elem={POS=ADVERB\_DEGREE SemProp=temp} Elem={POS= ADJECT}} →

{Elem={POS=ADJECT SemProp=\*{DEGREE=::temp}}} (…)

Elem – атрибут элемента, с его помощью обозначается линия в ОА-списке (описание предложения представляет собой ОА-список толкований слов, входящих в предложение). Левая часть продукции задает граф-шаблон, т.е. последовательность описаний двух слов, первое из которых – наречение степени, а второе – прилагательное). В host-графе находится подграф, совпадающий с графом-шаблоном из левой части продукции, то начинается преобразование графа: результат преобразования задается в правой части продукции. Согласно (…) фрагмент списка из двух лексем должен преобразования во фрагмент из одной лексемы (части речи «прилагательное», а ИК с описанием семантических свойств наречия степени, снабженное атрибутом «Degree» перемещается в описание семантических свойств прилагательного. Для описания такого перемещения применяется переменная temp. Знак «\*=» означает, что ИП после него конкатенируется к ИК, указатель которой находится в нагрузке ИП с атрибутом «SemProp» [Salibekyan, S. and Panfilov, P. Linguistic Processor Based on Object-Attribute Grammar. In Supplementary Proc.: The 6th International Conference on Analysis of Images, Social Networks, and Texts (AIST-2017), Moscow, Russia, July 27-29, 2017, CEUR Workshop Proceedings, ISSN 1613-0073, Vol-1975, pp.134-145, URL: <http://ceur-ws.org/Vol-1975/paper15.pdf>].

**Семантический анализ естественного языка (Natural Language Processing (NLP))**

Возможность создания семантической сети и ее преобразование с помощью ОА-грамматика позволяют на базе ОА создавать систему семантического анализа естественного языка (ЕЯ). На вход ОА системы семантического анализа ЕЯ (Natural Language Processing (NLP)) поступает анализируемый текст, описание слов (семантико-синтаксический словарь) и база знаний в виде набора ОА продукций для анализа предложения. На выходе – ОА семантическая сеть, описывающая смысл текста. Полученная в результате работы системы NLP, семантическая сеть может быть использована, например, в интернет-поисковой системе для семантического поиска информации.

Работа ОА NLP подразделяется на несколько стадий [Салибекян С.М. Объектно-атрибутный подход для семантического анализа естественного языка // Информационные технологии. 2021. Т. 27. № 5. С. 267-274.]. Многостадийный анализ естественного языка также применяется в теории Теория [I.A. Bolshakov, A.F. Gelbukh. The Meaning-Text Model: Thirty Years After J. International Forum on Information and Documentation, FID 519, ISSN 0304-9701, N 1, 2000.] предполагает анализ текста на нескольких уровнях, с постепенным переходом от одного уровня к другому: фонологического (уровня текста) и семантического (уровень смысла), поверхностно-морфологический, глубинно-морфологический, поверхностно-синтаксический, глубинно-синтаксический и семантический (уровень смысла). Однако данная теория практически не нашла применения на практике ввиду свой сложности и негибкости. ОА-подход от теории смысл<->текст отличается единообразием информационных конструкций на всех стадиях семантического анализа, а также наличием аппарат ОА-грамматика, позволяющим производить преобразование ОА семантической сети, используемой в процессе анализа языка, без опасения потери целостности данных в ней.

Итак, первый этап ОА анализа – это формирование списка толкований слов предложения (в ОА подходе анализа осуществляет по предложениям). Толкования расположены в семантико-семантическом словаре. Каждая запись словаря представляет собой ИК с описанием синтаксических свойств слова; в данной ИК обязательно находится ИП с атрибутом SemProp и с указателем на описание семантических свойств слова в нагрузке. ОА-система ищет по словарю слова из предложения формирует из найденных семантико-синтаксических описаний ОА-список, который на последующих стадиях анализа преобразуется во фрагмент семантической сети с описанием смысла текста. Пример фрагмента списка толкований слов приведен на рис. … .

Далее наступает происходит преобразование списка толкований слов в семантическую сеть за несколько этапов. Каждый этап соответствует анализ одной части речи или семантико-семантической конструкции. Если данная часть речи находится в списке толкований слов, то рядом ищется главное слово, и далее происходит склейка толкований слов, когда семантическое описание зависимого слова переносится в описание главного слова, и затем описание зависимого слова удаляется из списка толкований слов за ненадобностью. С каждым этапом анализа количество толкований в списке уменьшается. На заключительных этапах анализа производится разбор крупных семантико-синтаксических конструкций, таких как причастные и деепричастные обороты или предикативные части предложения. В конце анализа в списке толкований слов остается только один элемент, ссылающийся на семантическую сеть с описанием смысла предложения.

Связывание толкований предложений в единую семантическую сеть происходит с помощью тематического словаря. В него включаются описания объектов, на которые могут быть ссылки в следующих предложениях. Например, предложения «В лесу водились **белки**.» и «**Белки** были рыжего цвета.» связаны через слово «белки». При разборе первого предложения слово «белки» попадает в тематический словарь, а при анализе второго предложения система семантического анализа найдет толкование этого слова и тематическом словаре и создаст семантическую связь между толкованиями этих предложений. Структура ОА семантического анализа ЕЯ приведена на рис. … .



Рис. … . Состав ОА системы семантического анализа ЕЯ

С помощью ОА подхода решается проблема анализа многозначности толкований слов (полисемии). Так в тематическом словаре слово может быть представлено в виде списка его различных толкований. Тогда в список толкований слов попадает описанием сразу нескольких толкований для каждого слова (рис. …). В ходе анализа на определенном этапе анализа текста перед склейкой описаний слов проверяется их синтаксико-семантическое согласование. Если слова согласуются, то производится их склейка. Если же согласуются несколько толкований, то формируются толкования списки альтернативных толкований слов. Например, на рис … производится склайка двух слов с двумя толкованиями у каждого. Всего получается 4 возможных альтернативных ветвей толкования. На одной из ветвей происходит склейка толкований, остальные остаются в первоначальном виде. Если же на ветви альтернативного толкования обнаруживается несогласование толкований слов, то эта ветвь удаляется. Таким образом, во время анализа предложения происходит постоянное создание и удаление альтернативных ветвей толкования. Если в конце анализа остается только одно толкование всего предложения, то оно распознано однозначно; иначе существует несколько толкваний.

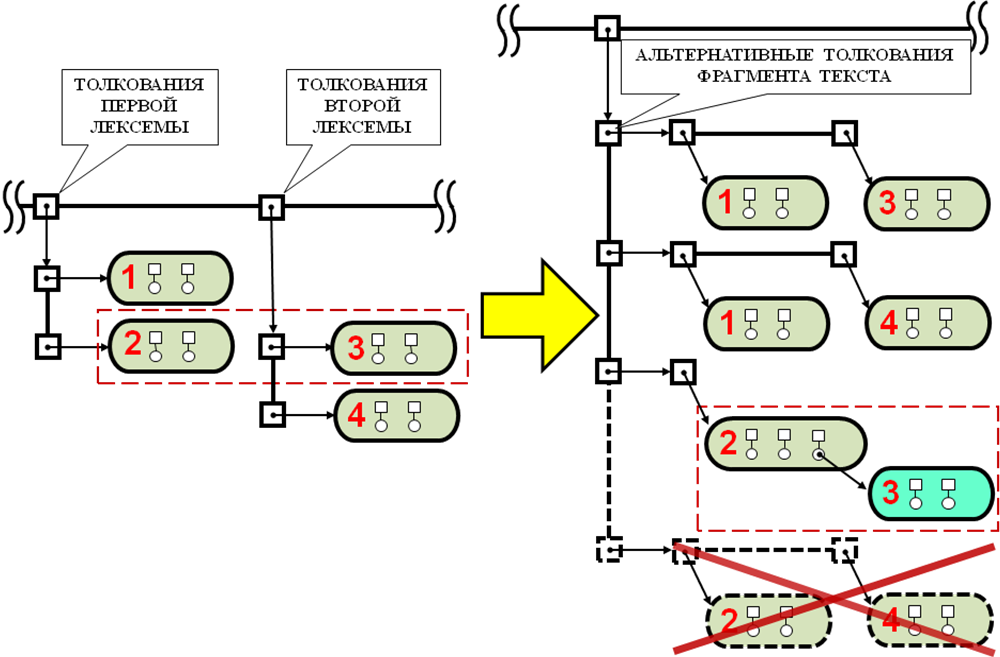


Рис. … . Обработка многозначности слов естественного языка

Для представления смысла текста была разработана структура семантической сети [С.М. Салибекян, С.Б. Петрова Методика анализа смысла текста с помощью семантической сети, содержащей нечеткие данные // Нейрокомпьютеры: разработка и применение, №4, 2016. p. 37-42]. На рис. … проиллюстрирована семантическая сеть для предложения «Он увидел машину через окно». На рис. … представлена схема расположения объектов, описанных в предложении.

 Рис. … . Семантическая сеть для описания пространственно-временнЫх отношений



Рис. … . Схема пространственно-временнЫх отношений

В предложении подразумевается три объекта: «наблюдатель» (*Person*), «окно» (*Window*) и «машина» (*Car*). Описание всех трех объектов присутствуют на уровне описания объектов. Каждый объект входит в множество (Set), состоящее только их этого объекта. Поэтому во всех множествах присуствует только один элемент («Element»), ссылающийся на описание элемента. Атрибут «Class» идентифицирует класс объекта, «Atr» - атрибут (свойство объекта). Свойства объектов также могут образовывать множество (например, «большой» и «зеленый»; «маленький» или «большой»). Поэтому первый подуровень уровня описания свойств и состояний объектов является описанием типа множество свойств. На втором подуровне расположены ИК, хранящие ссылки на элементы множества. В нашем случае у каждого объекта имеется по одному свойству или состоянию. Наблюдатель (Persone) находится в состоянии (Stage) наблюдения (Look), а окно и автомобиль в неизвестном состоянии (nil). Все объекты связаны с одним конфайном (confine). Под конфайном (от англ. confine – ограничивать) понимается описание пространственно-временнЫх отношений между объектами (или пространственно-временных ограничений). В данный конфайн входит три элемента, ссылки на которые находятся в ИК. Т.к. в данном случае в множестве важна последовательность элементов, указатель на ИК снабжен атрибутом SetOrder (упорядоченное множество). ZeroPoint – точка отсчета (начало координат). В нашем случае это – наблюдатель. AxisPoint – точка, задающая координатную прямую. В наше случае это – «окно», т.е. координатная прямая проходит от наблюдателя и проходит через окно. Direction – направление оси (в нашем случает прямое направление). Третьим (последним) элементом является «Машина».

Таким образом, ОА подход дает возможность производить семантический анализ текста с созданием семантической сети с его описанием. Методики ОА анализа имеет следующие преимущества:

- единообразное представление данных на всех этапах анализа, что сводит к минимуму преобразования информационных конструкций при переходе от одного этапа анализа к другому;

- существует инструмент ОА-грамматики, обеспечивающей описание преобразований как структуры семантической сети, так и информационного содержания в ней;

- ГТС обеспечивает целостность данных в семантической сеть при ее преобразовании;

- исходные данные, базу знаний, семантико-синтаксический словарь можно описать с помощью такого удобного инструмента, как ОА-язык;

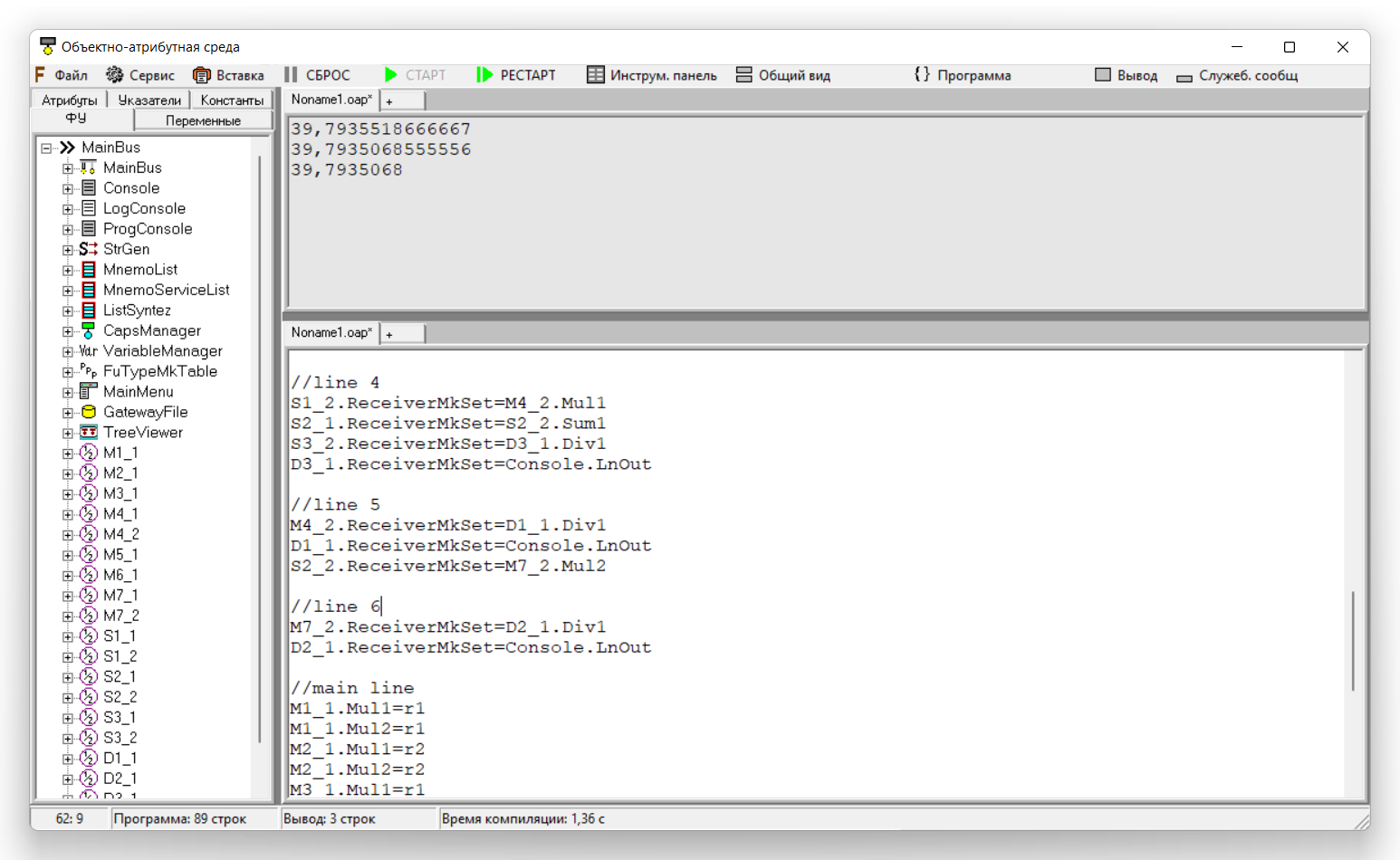
- ОА ВС, осуществляющая анализ текста, может быть реализована как программно, так и аппаратно.

**3 Среда объектно-атрибутного программирования и моделирования**

**3.1 Интерфейс среды программирования**

Лабораторный практикум выполняется с помощью среды объектно-атрибутного программирования и моделирования. В состав этой среды входят: текстовый редактор для ввода программы, компилятор ОА-языка программирования, средства для создания виртуальной ОА ВС, средства имитационного моделирования и другие инструментальные средства. С помощью среды можно создавать виртуальную реализацию ОА ВС, запускать на ней выполнение вычислительного процесса, а также проводить имитационное моделирование вычислительного процесса.

Интерфейс среды представлен на рисунке 8.



Инструментальная панель

Список созданных функциональных устройств

Консоль программы

Консоль вывода

Рис. 8. Интерфейс среды программирования и моделирования Millicom

Среда программирования и моделирования, с помощью которой выполняется лабораторный практикум имеет четыре основных поля. Консоль вывода принимает сообщения от программы для пользователя. Консоль программы позволяет вводить и редактировать текст ОА-программы. Инструментальная панель представляет оператору необходимые инструменты для написания ОА-программы и отслеживания вычислительного процесса в виртуальной ОА ВС. Сообщения об ошибках, возникших во время компиляции программы выводятся в консоль логов. Консоль при возникновении ошибок появляется низу окна среды ОА программирования и моделирования.

Для правильной работы программы во время выполнения лабораторного практикума необходимо, чтобы консоль программы имела тип «Root tab». В данном режиме программа из консоли перезапускается, т.к. производится сброс среды и затем заново осуществляется интерпретация программы. Для смены типа консоли необходимо нажатием правой кнопки мыши на вкладке с названием файла, отраженного на консоли, вызвать контекстное меню и выбрать пункт «Root tab». После этого кнопка «СТАРТ» верхнего меню станет недоступной, а кнопка «РЕСТАР» доступной для нажатия. С помощью данных кнопок осуществляется компиляция и запуск ОА-программы из текущего окна программы.

Панель программы может иметь несколько вкладок. Новую вкладку консоли можно открывать с помощью знака «+». Новая консоль также создается при открытии нового файла программы. Открыть файл можно с помощью пункта меню «Файл->Открыть» или с помощью комбинации клавиш «Ctrl-O». Также имеется пункт меню «Файл->Последние файлы», где имеется список последних открытый в среде программирования и моделирования файлов. Закрыть программную вкладку можно с помощью пункта «Close» контекстного меню, вызванного над названием файла в программной консоли (рис. 9). Сохранить текст из вкладки можно с помощью пункта меню «Файл->Сохранить», или горячей клавиши «Ctrl-S». Для записи файла под другим именем применяется «Файл->Сохранить как…».

Консоль вывода также может содержать несколько вкладок. Работа со вкладками консоли вывода аналогично программной панели.

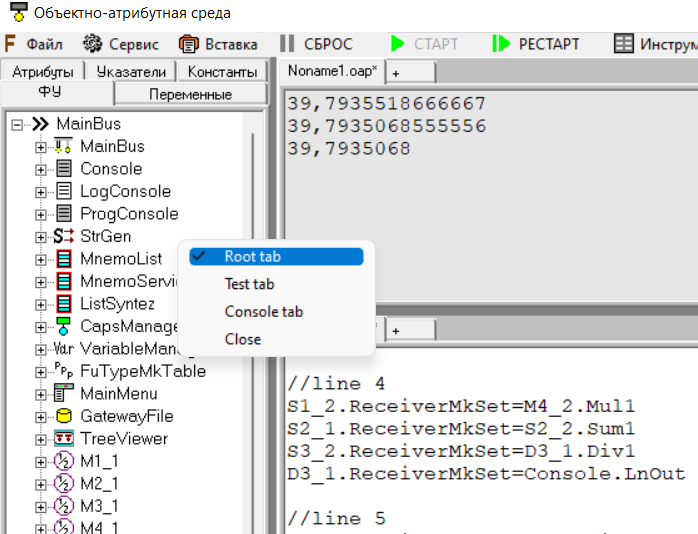


Рис. 9. Настройка режима «Root tab» для консоли среды

Консоль сообщений (служебная консоль) об ошибках компиляции программы. Обычно это окно скрыто, но оно появляется в низу окра среды при возникновении ошибок. Скрыть или показать данное поле можно с помощью кнопки «Служеб. сообщения» в верхнем меню. При возникновении ошибки в служебную консоль выводится строка программы, где была найдена ошибка, указание номера этой строки, а также сообщение компилятора о типе ошибки. Если совершить двойной щелчок мыши на данной консоли, то консоль скроется, а текстовый курсор перейден на ту строку в консоли программы, где была обнаружена ошибка.

В среде имеется возможность сохранения и считывания файла ОА-программы с помощью пунктов меню Файл->Открыть и Файл->Сохранить и Файл->Сохранить как… Для быстрого открытия недавно редактируемых файлов программы присутствует пункт меню Файл->Последние файлы. Запись программы происходит для консоли, на которая в данный момент получила фокус ввода. Закрытие вкладки осуществляется с помощью пункта «Close» контекстного меню, вызываемого из вкладки с названием файла (рис. 9).

Инструментальная панель включает в себя несколько вкладок. Вкладка «ФУ» отображает список виртуальных ФУ, созданных в среде программирования и моделирования (рис. 10). ФУ представляют собой виртуальные устройства, поэтому их можно создавать и выгружать из памяти в любой момент. Создание ФУ осуществляется с помощью команды ОА-языка NewFU{параметры создаваемого ФУ}. При нажатии на знак «+» радом с именем ФУ в данной вкладке разворачивается список атрибутов милликоманд, с которыми работает ФУ; двойное нажатие на мнемонику милликоманды вызывает появление название ФУ и мнемоники милликоманды в окне программы на месте, где находится курсор (таким образом ускоряется написание ОА-программы).

****

Рис. 10 Вкладка «ФУ» панели инструментов

Вкладка «Атрибуты» иллюстрирует созданные в ОА-среде информационные атрибуты. Вкладка «Указатели» - соответственно, указатели, «Константы» - именованные константы, «Переменные» - мнемоники переменных. Перенос мнемоники из панели в программную вкладку на место курсора осуществляется с помощью двойного нажатия мышки на соответствующей мнемонике.

На верхнем меню ОА среды имеются кнопки для настройки вида среды программирования (рис. 8). Кнопка «Инструментальная панель» показывает или скрывает левую инструментальную панель. «Общий вид» показывает одновременно программную консоль и консоль вывода. «Программа» активирует только программную панель, «Вывод» - только консоль вывода. Кнопка «Служебные сообщения» служит для показа/скрытия служебной консоли.

Пункт меню «Вставка» предоставляет возможность быстрой вставки наиболее часть используемых конструкций ОА-языка. В меню также приведены горячие клавиши для осуществления быстрой вставки.

Пункт меню «Сервис» дает возможность перехода в англоязычный интерфейс, настройку автозагрузки файлов программы при запуске ОА-среды, поиска фрагмента текста в файле, поиска и замены текста, настройки шрифтов выводной и программной консолей, записи выводной консоли в файл и т.д.

**3.2 Имитационное моделирование с помощью среды программирования и моделирования**

В состав среды программирования также входят средства имитационного моделирования вычислительного процесса. Моделирование вычислительного процесса ведется с помощью событийной модели [Jerry Banks, John S. Carson II, Barry L Nelson, David M. Nicol Discrete-Event System Simulation] и модели процессной модели Кана (KPN) [Ключев А.О., Кустарев П.В., Ковязина Д.Р., Петров Е.В. Программное обеспечение встроенных вычислительные систем. – СПб.: СПбГУ ИТМО, 2009. – 212 с.]. Согласно модели KPN, если милликоманда приходит к ИУ, занятому обработкой другой, то оно становится в очередь ожидания освобождения исполнителя. В среде программирования и моделирования присутствуют специализированные ФУ для моделирования вычислительного процесса: Scheduler (планировщик ресурсов) и Eventser (контроллер событий в ВС). Процесс моделирования проиллюстрирован на рис. 11.

Вычислительное устройство 1

Вычислительное устройство n

t

Событие 1 (текущее)

Событие k

Очередь на ожидание вычислительных ресурсов

Исполнительные устройства (процессорные ядра)

Шлюз

Рис. 11. Методика моделирования data-flow вычислительной системы

ФУ «Планировщик» эмулирует распределение ресурсов на вычислительном узле с общей памятью. Основной характеристикой узла является количество параллельно работающих ИУ (например, процессорных ядер). ФУ в таком узле могут находится в двух состояниях: активном и ожидании. В первом им предоставляется исполнительное устройство для вычислений, во втором для ФУ не хватает вычислительных ресурсов, и оно ожидает их освобождения. Контроллер событий контролирует последовательность наступающих во время выполнения вычислительного процесса событий. Длительность вычислительного процесса определяется наличием вычислительных ресурсов, коэффициентом параллелизма алгоритма, а также длительностью вычислительных операций (устанавливается во время настройки имитационной модели).

Scheduler управляется с помощью следующих МК:

NCoresSet: Установить число исполнительных устройств (процессорных ядер) в ВС

ScheduleTimeSet: Установить время, затрачиваемое планировщиком для обработки запроса

EventserContextSet: Установить контекст Eventser

CoreCountPopMk: Выдать милликоманду с количеством занятых ядер

SchedulerMkQueuePopMk: Выдать милликоманду с количеством милликоманд, находящихся в очереди к исполнению на ИУ

SchedulerMkQueueMaxPopMk: Выдать милликоманду с максимальным количеством милликоманд, находящихся в очереди к исполнению на ФУ

Основные МК для Eventser:

- EventWaitSet: Добавить ожидание событие

- ModelReset: Сброс параметров моделирования

- CurrentTimePopMk: Выдать милликоманду с текущим модельным временем

- ParallelFactorPopMk: Выдать милликоманду с коффициентом параллелизма

- EventsCountPopMk: Выдать милликоманду с числом произошедших событий

Привязка ФУ к планировщику осуществляется с помощью МК SchedulerContextSet. Пример, привязки ФУ Mul1 к Scheduler.

Scheduler.ContextPopMk=Mul1.SchedulerContextSet

Параметры имитационного моделирования для исполнительных ФУ задаются с помощью специального набора МК. Для каждого типа ФУ такой набор индивидуален. Например, для ФУ «Потоковое АЛУ», они следующие:

- SumTimeSet – установить время операции суммирования;

- RegLoadTimeSet – установить время считывания операнда;

- SubTimeSet – установить время операции вычитания;

- MulTimeSet – установить время операции умножения;

- DivTimeSet – установить время операции деления.

И т.д.

В состав имитационной модели может входить достаточно много ФУ с однотипными настройками моделирования. И для того, чтобы описание настроек сократить, применяется шаблонное создании ФУ. Суть этого приема в том, что вначале создается ФУ (эталонное ФУ) и для него осуществляются все настройки, а затем в однотипные ФУ переписывается контекст эталонного ФУ. Для того, чтобы создать ФУ по шаблону, необходимо в МК создания ФУ в вместо мнемоники типа поместить указатель на эталонное ФУ. Приведем фрагмент кода из приложения В:

NewFU={Mnemo="Mul1" FUType=FUStreamFloatAlu} //Эталонное ФУ

Scheduler.ContextPopMk=Mul1.SchedulerContextSet // Задать ссылку на планировщик вычислений

// Настройки моделирования для эталонного ФУ

Mul1.SubTimeSet=2

Mul1.MulTimeSet=9

Mul1.DivTimeSet=10

Mul1.SumTimeSet=2

Mul1.SqrtTimeSet=8

Mul1.RegLoadTimeSet=0.2

ContextTemplate=nil

Mul1.ManualModeSet=true

Mul1.ContextPop=***ContextTemplate*** // Создать указатель на эталонное ФУ

NewFU={Mnemo="M1\_1" FUType=***ContextTemplate***} // Создание ФУ по шаблону

Пример настройки имитационного моделирования вычислительного процесса для 1-го лабораторного практикума можно посмотреть в приложении В.

**3.3 Визуализация результатов имитационного моделирования**

В состав среды ОА программирования и моделирования входит ФУ, осуществляющее визуализацию процесса работы ОА ВС в виде диаграммы Ганта [Калугин Владимир Евгеньевич Качество в проектировании и разработках: методические указания по выполнению курсового проекта [Электронный ресурс] / Е.В. Калугин, В.А. Осит – Омск: СибАДИ, 2017 – 40 с. URL: https://portal.sibadi.org/pluginfile.php/101557/mod\_resource/content/1/esd395.pdf , Рассел, Джесси Диаграмма Ганта / Джесси Рассел. – М.: VSD, 2012. – 591 c.]. Настройка визуализации осуществляется следующим образом. Для начала создается специализированное ФУ для вывода диаграммы Ганта

NewFU={Mnemo="Gant" FUType=FUGant}

Затем в ФУ Ганта записывается ссылка на переменную, где Eventser хранит текущее модельное время.

Eventser.CurrentTimePointPopMk=Gant.CurrentTimeRefSet

Затем для Eventser настраивается программа вывода результатов (Gant.EventSet - МК установки события для диаграммы Ганта):

Eventser.OutProgSet={Eventser.EventCapsPopMk=Gant.EventSet}

И под конец необходимо установить программу запроса события для Eventser настраивается программа вывода результатов (Eventser.EventRequestCapsPopMk – МК выдачи ИК с описанием последнего события):

Eventser.EventRequestProgSet={Eventser.EventRequestCapsPopMk=Gant.OperandsSet}

Результат визуализации работы ОА ВС можно увидеть на рис … .

**4. Практическая часть**

**4.1 Работа 1: Моделирование параллельного вычислительного процесса с альтернативными ветвями**

**4.1.1 Задание для работы 1**

Необходимо разработать параллельный алгоритм с параллельными ветвями для решения одной из геометрических задач (варианты 1-30) с помощью алгоритма с альтернативными ветвями вычислений. ***В исходное задание разрешается добавлять любые исходные данные (длины сторон, углы, радиусы и т.д.) по на усмотрение обучающегося*.** Формулы вычислений заданной характеристики геометрической фигуры можно получить (вывести) или взять из других источников. Алгоритм должен иметь не менее двух (желательно 3) альтернативных ветвей для вычисления результата. Последовательность выполнения работы описана ниже после вариантов заданий.

Требования к работе:

- потоковый граф должен включать в себя не менее 20-ти вершин (имеются ввиду вершины-операции, обозначенные на схеме с помощью кружка);

- алгоритма должен содержать не менее 2-х альтернативных ветвей (рекомендуется 3).

В работа оценивается:

- количество вершин-операторов в потоковом графе;

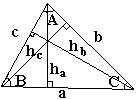
- наличие таблицы программ функциональных устройств;

- Количество параллельных ветвей.

**4.1.2 Варианты заданий для работы 1**

1. Площадь треугольника,

где a, b, c - стороны треугольника;



A, B, C - углы треугольника;

ha, hb, hc - высоты треугольника.

2. Площадь параллелограмма,

где a,b – стороны параллелограмма;

A

B

C

D

b

α

β

γ

σ

O

a

с

d

χ

δ

c,d – диагонали параллелограмма;

α,β – углы, образованные пересечением диагоналей;

γ, σ, χ, δ - углы, образованные стороной и диагональю.

3. Площадь равнобокой трапеции,

где a,c – основания;

AA

DB

СC

B

ada

fb

α

β

γ

bс

O

Ed

hf

gb

cda

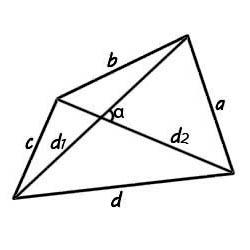
b – боковая грань;

h – высота;

f,g – отрезок от угла трапеции до пересечения диагоналей;

α,β – углы пересечения диагоналей;

γ – угол пересечения основания и диагонали.

4. Площадь четырехугольника

a,b,c,d – стороны четырехугольника;

α – острый угол между диагоналями.

5. Площадь поверхности усеченного конуса,

где a – боковая грань;

r,d – радиус и диаметр основания;

O'A

dB

rC

a

d

Bd

AO

Ph

OA

hC

r' d'

Cd

Dd

r',d' – радиус и диаметр верхней части пирамиды;

h – высота;

P, P' – периметр основания и верхней части;

S, s – площади основания и верхней части пирамиды.

6. Объем усеченного конуса,

r,d – радиус и диаметр основания;

O'A

dB

rC

a

d

Bd

AO

Ph

OA

hC

r' d'

Cd

Dd

r',d' – радиус и диаметр верхней части пирамиды;

h – высота;

P, P' – периметр основания и верхней части;

S, s – площади основания и верхней части пирамиды.

7. Площадь поверхности цилиндра,

где r,d – радиус и диаметр основания;

A

B

AC

h

d

r

O'

P

b

P

D

C

O

r

h – высота;

b – диагональ цилиндра;

P – периметр основания;

S – площадь основания.

8. Объем цилиндра (см. рис выше).

9. Площадь поверхности цилиндра в основании которого лежит эллипс

r – малый радиус эллипса;

A

B

AC

h

D

r

O'

P

b

P

D

C

O

d

R – большой радиус эллипса;

h – высота;

b – малая диагональ цилиндра;

B – большая диагональ цилиндра.

P – периметр основания;

S – площадь основания.

10. Объем цилиндра в основании которого лежит эллипс (см. рис выше).

11. Площадь поверхности усеченной пирамиды с квадратом в основании,

h – высота;

а – боковая грань;

b – грань квадрата, лежащего в основании;

A1

C

a

d

b

O

h

D

B

O'

b'

d'

d – диагональ квадрата, лежащего в основании;

S – площадь квадрата, лежащего в основании пирамиды.

12. Площадь поверхности усеченной пирамиды с квадратом в основании (см. рис выше).

13. Площадь поверхности усеченной пирамиды с треугольником в основании.

h – высота;

A

h

d

B

O

O'

C

b

a

B'

b

B'

C'

A'

а – боковая грань;

b,с – грани треугольника, лежащего в основании;

d – медиана треугольника, лежащего в основании;

S – площадь треугольника, лежащего в основании пирамиды.

14. Объем поверхности усеченной пирамиды с треугольником в основании (см. рис выше).

15. Объем призмы с трапецией в основании

O

AA

DB

СC

B

ada

fb

α

β

γ

bс

O

Ed

hf

gb

cda

B’dsdsd’’’’s

A’

D’

C’

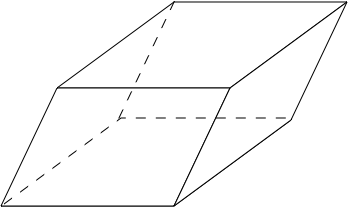
H

16. Площадь поверхности призмы с трапецией в основании (см. рис выше).

O

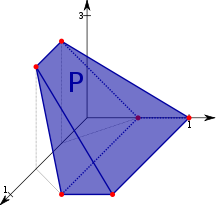
17. Объем косого параллелепипеда

O



18. Площадь поверхности косого параллелепипеда (см. рис выше).

19. Объем фигуры



20. Площадь поверхности фигуры (см. рис выше).

21. Объем фигуры



22. Площадь поверхности фигуры (см. рис выше).

23. Объем фигуры



24. Площадь поверхности фигуры (см. рис выше).

25. Объем шестигранной призмы



26. Площадь поверхности шестигранной призмы (см. рис выше).

27. Объем двусторонней пирамиды (в основании ромб)



28. Площадь поверхности двусторонней пирамиды (в основании ромб) (см. рис выше).

29. Объем двусторонней пирамиды (в основании равнобедренный треугольник)



30. Площадь поверхности двусторонней пирамиды (в основании равнобедренный треугольник) (см. рис выше).

**4.1.3 Этапы выполнения работы 1**

**1**. Найти несколько формул (не менее 2-х) для решения поставленной геометрической задачи. Начертить графы алгоритмов, описывающие все формулы нахождения площади треугольника.

**2.** Составить потоковые графы для каждой из альтернативных формул.

**3**. Составить общий граф программы по всем формулам. Составить таблицу программ для ФУ.

**4**. Ознакомиться со ОА-средой программирования и моделирования.

**5**. Составить и отладить вычислительную программу в среде ОА-программирования и моделирования для решения вычислительной задачи с альтернативными вычислительными ветвями.

**5**. Добавить в программу описание средств имитационного моделирования ВС. Определить коэффициент параллелизма, вывести диаграмму Ганта, реализуемую с помощью встроенных инструментальных средства среды.

**6**. Выбрать оптимальное количество процессорных ядер для решения вычислительной задачи.

7. Прогон программы по каждой из альтернативных вычислительных ветвей

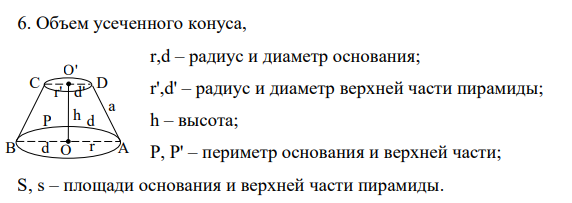
8. Запуск программы с произвольным временем поступления исходных данных.

9. Предлагается оценить, как бы данный алгоритм мог быть реализован в классической парадигме, насколько бы возросла сложность его реализации.

**4.1.4 Пример выполнения работы 1**

**Исходные данные:**

***Задание:*** найти объем усеченного конуса.



**Длительность работы ФУ:**

Сумматор – 2 τ;

Умножитель – 9 τ;

Делитель – 10 τ

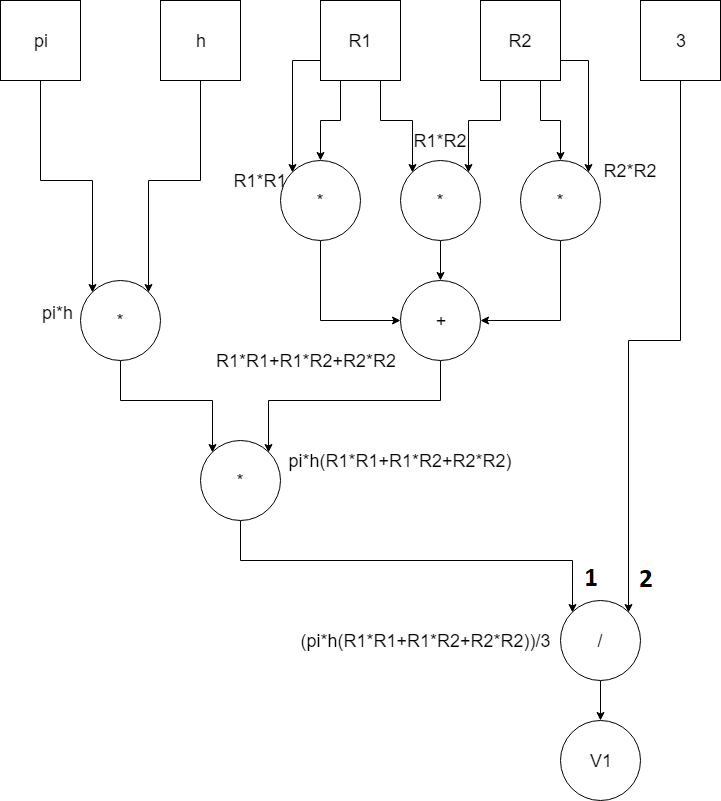
Тригонометрич. Блок – 10 τ;

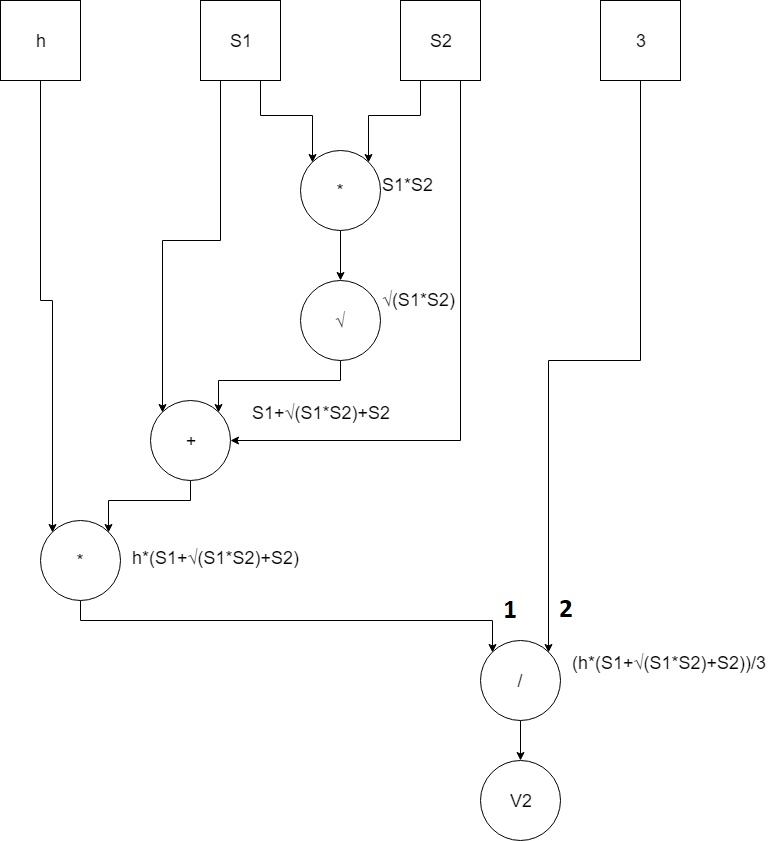
Коренер – 8 τ.

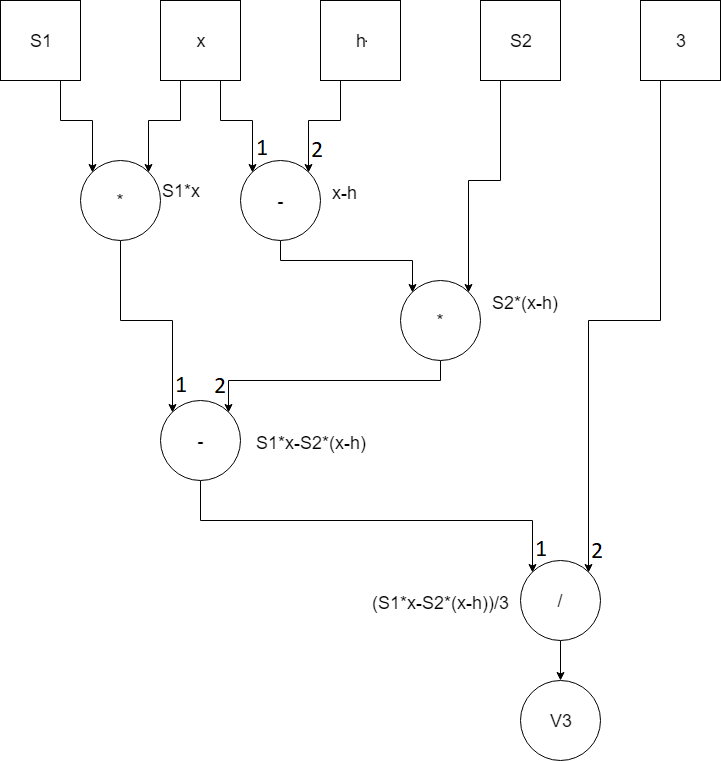
**Формулы для вычисления объёма усеченного конуса:**



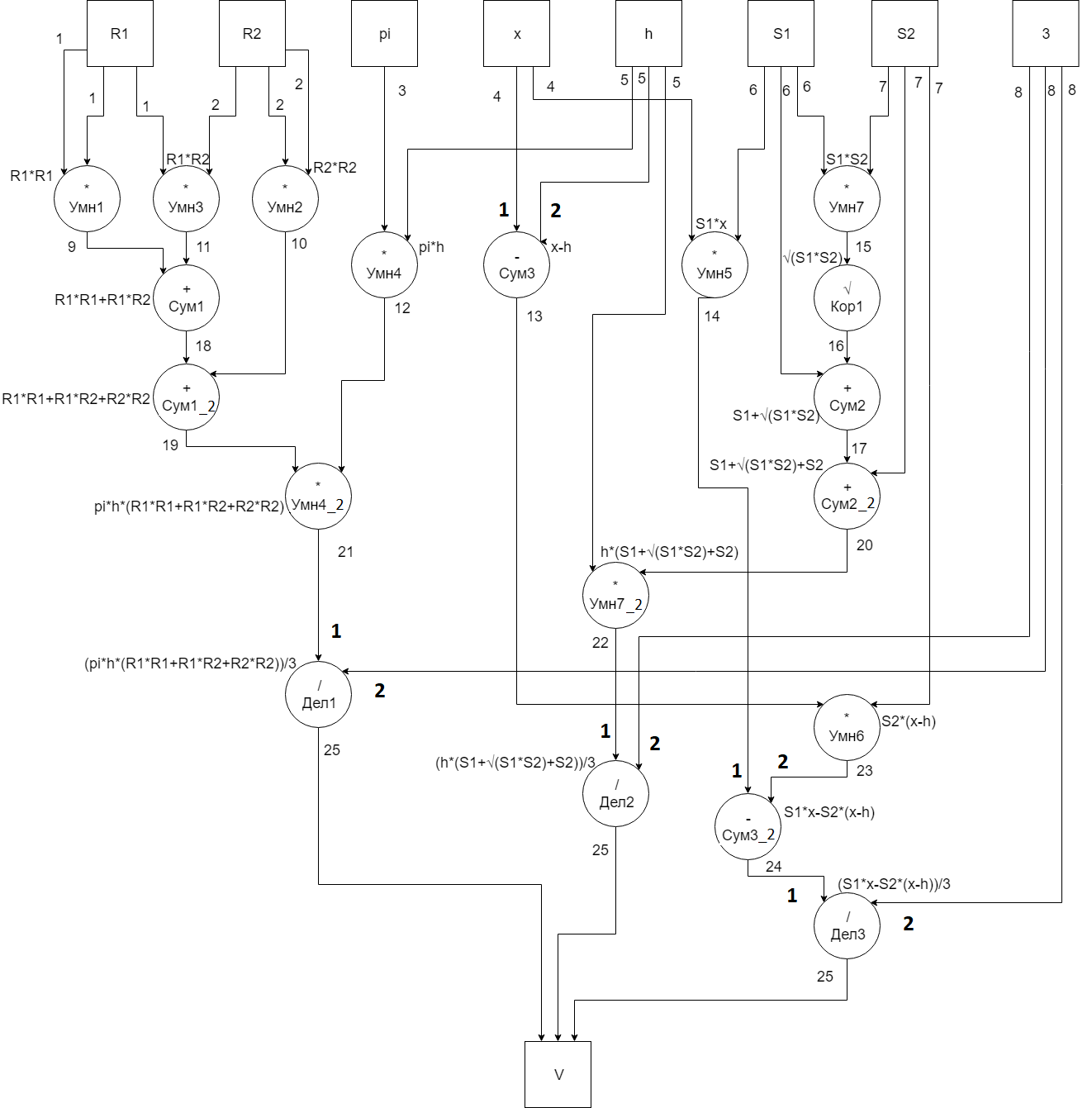
**Графы, описывающие формулы нахождения объёма усечённого конуса**







**Граф общего алгоритма нахождения объёма усечённого конуса**



**Таблица программ функциональных устройств**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Устройство | Ярлыки входных данных | Операция | Комментарии | Ярлык выходных данных |
| Умножитель 1 | 1 | ^2 | R1\*R1 | 9 |
| Умножитель 2 | 2 | ^2 | R2\*R2 | 10 |
| Умножитель 3 | 1,2 | \* | R1\*R2 | 11 |
| Умножитель 4 | 3,5 | \* | pi\*h | 12 |
|  | 12,19 | \* | pi\*h\* (R1\*R1+R1\*R2+R2\*R2) | 21 |
| Умножитель 5 | 4,6 | \* | S1\*x | 14 |
| Умножитель 6 | 7,13 | \* | S2\*(x-h) | 23 |
| Умножитель 7 | 6,7 | \* | S1\*S2 | 15 |
|  | 5,20 | \* | h\*(S1+√(S1\*S2)+S2) | 22 |
| Сумматор 1 | 9,11 | + | R1\*R1+R1\*R2 | 18 |
|  | 10,18 | + | R1\*R1+R1\*R2+R2\*R2 | 19 |
| Сумматор 2 | 6,16 | + | S1+√(S1\*S2) | 17 |
|  | 7,17 | + | S1+√(S1\*S2)+S2 | 20 |
| Сумматор 3 | 4,5 | - | x-h | 13 |
|  | 14,23 | - | S1\*x-S2\*(x-h) | 24 |
| Коренер 1 | 15 | √ | √(S1\*S2) | 16 |
| Делитель 1 | 8,21 | / | (pi\*h\*  (R1\*R1+R1\*R2+R2\*R2)) /3 | 25 |
| Делитель 2 | 8,22 | / | (h\*(S1+√(S1\*S2)+S2))/3 | 25 |
| Делитель 3 | 8,24 | / | (S1\*x-S2\*(x-h))/3 | 25 |

**Создание миллипрограммы решения вычислительной задачи**

Имитационное моделирование ВС проводится с помощью среды ОА-программирования и моделирования (документация к среде программирования и моделирования приведена в приложении А), которая позволяет создавать виртуальную копию компьютерной ВС и проводить ее исследование с целью определения оптимальных параметров ВС. В среде применяется специальный язык программирования (ОА-язык), с помощью которого можно не только задавать вычислительный алгоритм, но и конфигурировать модель ВС, задавать входные данные, настраивать программную модель.

Моделирование можно проводить по следующей схеме.

1. Построение потокового графа решения вычислительной задачи.

2. Составление ОА-программы для решения вычислительной задачи в среде ОА-программирования и её отладка.

3. Дополнение ОА-программы для имитационного моделирования: вводится конфигурация ВС (вычислительные узлы, связи между вычислительными узлами, конфигурация вычислительных узлов, настройки вывода результатов моделирования и т.д.).

4. Проведение серии прогонов имитационной модели с целью определения оптимальной конфигурации ВС.

**Составление ОА-программы для решения вычислительной задачи**

Первый этап составления ОА-программы – создание виртуальных ФУ, для решения задачи. Для этого необходимо переименовать все вершины потокового графа алгоритма (имена вершин будут являться именами ФУ, созданных в среде ОА-программирования). Имена ФУ необходимо подписывать непосредственно на потоковом графе, т.к. это сделает дальнейшее выполнение работы более удобным. Создание ФУ осуществляется с помощью милликоманды NewFU (горячая клавиша в среде ОА-программирования - ctrl+d). В капсуле, передаваемой с данной милликомандой могут находится информационные пары (ИП) со следующими атрибутами: Mnemo – название создаваемого ФУ, FUType – тип создаваемого ФУ, Hint – всплывающая подсказка, которая появляется при наведении курсора на название ФУ в среде ОА-программирования (данный параметр является необязательным).

Будем придерживаться следующего правила именования ФУ: сумматор и вычитатель обозначается Sum, умножитель – Mul, делитель Div, ФУ для возведения числа в квадрат – Sqr, для извлечения квадратного корня – Sqrt, ФУ для получения синуса, косинуса, тангенса, котангенса – Sin, Cos, Tg, Ctg соотвественно. Далее за названием, отображающем функционал ФУ, будет следовать порядковый номер ФУ: например, Sum1, Sum2 и т.д.

Также можно применять и другую стратегию именования ФУ. Давать имя начиная со слова Dev (от англ. Device) и далее выставлять через знак подчеркивания две числа: первое – номер вычислительной ветки, второе – порядковый номер ФУ в ветке. Те ФУ, которые участвуют в вычислениях сразу на нескольких вычислительных ветках при именовании могут получать либо номер одной из ветвей, либо именовать особым образом.

Для выполнения работы нам будет необходим один тип ФУ – FUStreamFloatAlu (потоковое арифметико-логическое устройство с плавающей точкой.).

Например, создание ФУ с именем Mul1 с типом FUStreamFloatAlu и всплывающей подсказкой «Первый умножитель» будет выглядеть так:

NewFU={Mnemo="Mul1" FUType=FUStreamFloatAlu Hint="Первый умножитель"}

В нашем примере нам будет необходимо создать 22 ФУ с соответствующими именами (мнемониками) и типом FUStreamFloatAlu.

Далее нам необходимо описать константы необходимые нам для вычислений. Константы задаются с помощью знака «#»: перед знаком помещается мнемоника константы, после – значение. Например, a#10. В нашем примере нам будут необходимы следующие константы: a – боковая грань, b – диагональ, H – высота, b – диагональ, P – периметр основания, S – площадь основания цилиндра:

a#10 b#10 H#10 P#10 R#10 S#10

Константы необходимо подбирать таким образом, чтобы она удовлетворяли условиям решения поставленной геометрической задачи. Константа Пи является встроенной в среду ОА и обозначается Pi.

Для того, чтобы ФУ были созданы в среде ОА-программирования необходимо запустить ОА-программу на выполнение. *Примечание: не забудьте сделать вкладку с ОА-программой корневой (Root)*. После запуска ОА-программы имена ФУ должны быть отображены в панели ФУ, а константы – в панели констант.

Следующим этапом является описание поступления исходных данных для ФУ и обмен данными между ФУ. ОА-программа, соответствующая разбираемому примеру приведена в приложении Б.

**Рекомендации по написанию ОА-программы**

К программе предъявляются следующие требования: не менее 20 вершин в потоковом графе (более 30 вершин дает дополнительный балл), не менее двух альтернативных вычислительных ветвей (лучше не менее трех), все выходные данные с различных вычислительных веток должны совпадать между собой.

Рекомендации по написанию программы. Создание программы лучше производить в следующей последовательности:

1. Объявить все ФУ, необходимые для вычислений.
2. Объявить все необходимые константы.
3. Ввести все исходные данные (раздел «исходные данные» в приложении 1). На потоковом графе исходные данные обозначаются прямоугольниками – и следует прописать все милликоманды к тем ФУ, к которым приходят стрелки из прямоугольников независимо от того, на каком уровня графа они находятся.
4. Затем производить поэтапное программирование информационных пересылок между ФУ (раздел «Описание пересылок данных»). Раздел описания пересылок находится между разделами описания констант и исходных данных. Во избежание совершения ошибок лучше всего производить программирование пересылок по ярусам потокового графа (ярус – вершины имеющие одинаковый максимальный путь от основания графа (т.е. от вершин, которые представляют исходные данные). При программировании каждого ФУ желательно выводить результаты его работы на консоль. Для этого можно воспользоваться милликомандой ИмяФУ.ReseaverMkSet=Console.LnOut. Например, для проверки того, что на ФУ Mul4 (приложение Б) приходят все операнды, надо записать M1.ReceiverMkSet=Console.LnOut. После запуска ОА-программы результат вычислений Mul4 должен появиться в консоли вывода. После этого меняем M1.ReceiverMkSet=Console.LnOut на M1.ReceiverMkSet=Sum1.Sum, т.е. меняем потребителя для M1 на его настоящего потребителя и приступаем к описанию пересылок других ФУ.

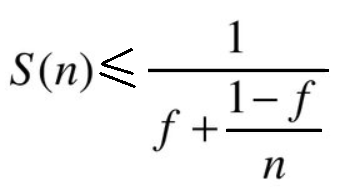
**Исследование модели параллельной dataflow-системы**

После того, как была реализована имитационная модель dataflow-системы, необходимо произвести ее исследование. Пример модификации программы для моделирования приведен в приложении В. Для моделирования необходимо скопировать заголовок программы для настройки моделирования, заменить в милликоманде создания ФУ

Исследование состоит из трех частей, которые можно условно именовать как: «Нахождение оптимального количества вычислительных ядер», «Исследование вычислительных ветвей » и «Исследование вычислительной системы при приходе исходных данных в случайные моменты времени».

**Нахождение оптимального количества вычислительных ядер.**

Коэффициент параллелизма есть отношение времени выполнения программы к времени выполнения программы в последовательном стиле. Или, иными словами, сколько вычислительной мощности (в нашем случае среднее количество процессорных ядер) в среднем используется во время выполнения вычислительного процесса. На коэффициент параллелизма влияют два фактора – количество ресурсов (процессорных ядер) в вычислительной системе и коэффициент параллелизма алгоритма (т.е. сам алгоритм может не давать возможности его распараллеливания). На этот случай имеется формула Амдала, описывающая зависимость коэффициента параллелизма в зависимости от данных двух параметров [Воеводин В.В., Воеводин Вл. В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002. ­­– 608 с.]:



где

*n* – количество вычислителей в составе вычислительной системы

*f* – коэффициент параллелизма алгоритма (*f*=1 – полностью последовательный алгоритм, f=0 – полностью параллельный алгоритм)

Исследование вычислительной системы проводится следующим образом. С помощью милликоманды Scheduler.NCoresSet устанавливается количество ядер в вычислительной системе. Далее после запуска программы в окно вывода выдается модельное время выполнения программы. Далее необходимо построить график зависимости времени выполнения программы от количества вычислительных ядер системы. Построение графика необходимо завершить в тот момент, когда с увеличением количества ядер на 1 время выполнения программы будет изменяться менее чем на 10% (см. рис.)



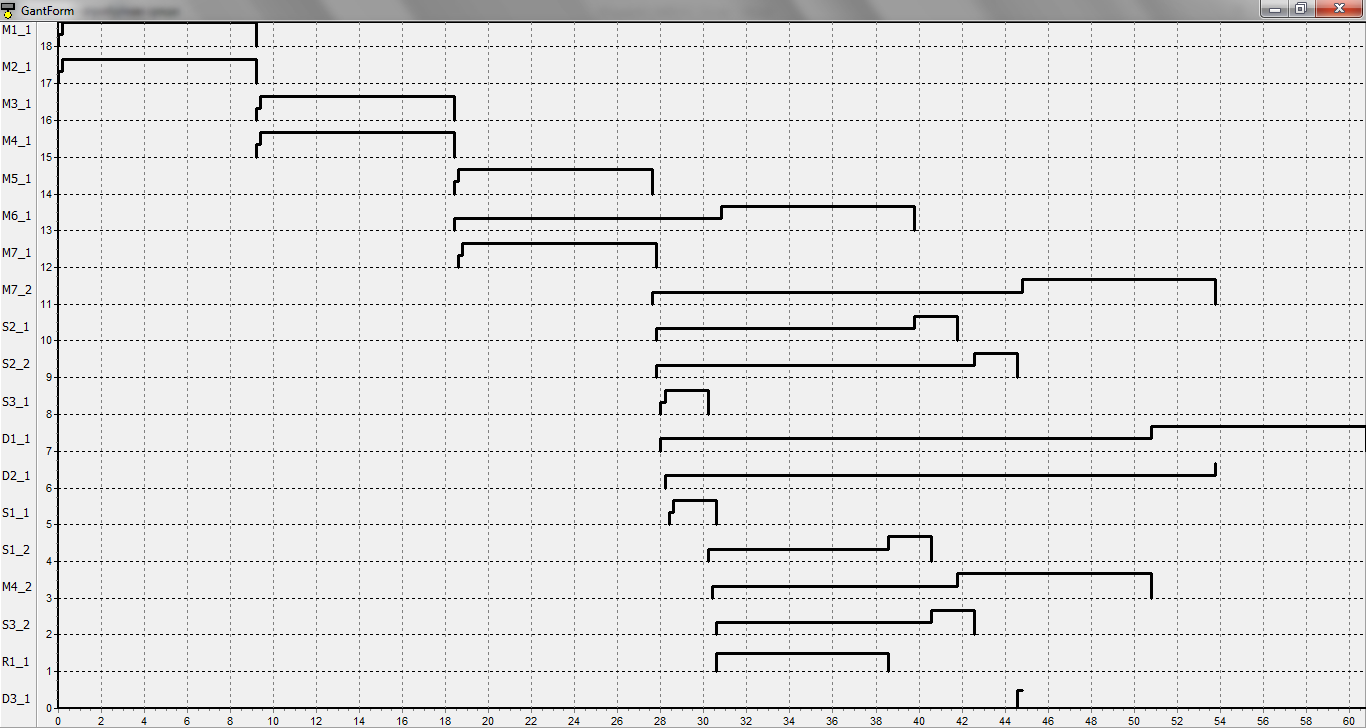
Судя по рисунку, увеличение количества ядер в составе вычислительной системы более чем на 5 нерационально.

**Исследование вычислительных ветвей.**

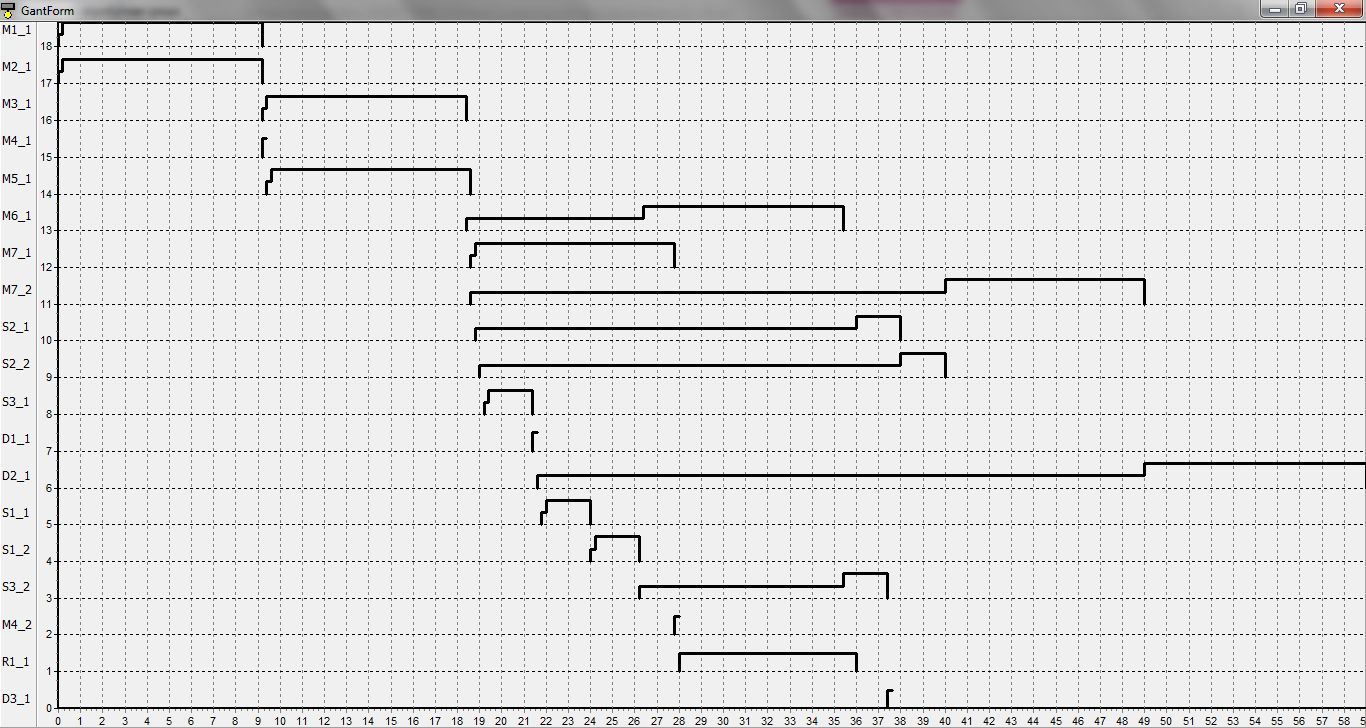
Данное исследование необходимо для определения скорости вычисления для каждой вычислительной ветви алгоритма. Для такого исследования необходимо прервать выполнение моделирования после того, как будет получен результат вычисления по вычислительной ветви с помощью милликоманды Eventser.Stop. Для этого необходимо дополнить каждую милликоманду вывода результата вычисления милликомандой Eventser.Stop. Например, Mul2\_2\_ReceiverMkSet=Console.LnOut нужно заменить на Mul2\_2\_ReceiverMkSet=Console.LnOut=Eventser.Stop. Этим буде обеспечено завершение процесса моделирования при получении первого результата. Количество ядер для данного исследования наиболее оптимально оставить равным 2 или 3.

После запуска теста смотрим, на каком из ФУ остановится процесс моделирования – это то ФУ, на котором закончился вычислительный процесс. В нашем случае – это ФУ D3\_1. Далее необходимо затормозить выполнение этой ветви алгоритма. Для этого необходимо по потоковому графу отследить, от каких входных данных зависит данная вычислительная ветвь (необходимо контролировать, чтобы этот входной операнд не влиял на другие вычислительные ветки). Для вычислительной задачи, приведенной в примере, входной операнд 3 влияет только на данную вычислительную ветку. Поэтому задержим поступление данного операнда, установив время его выдачи равным 100:

Eventser.EventWaitSet={ TimeAtr=100 Mk={D3\_1.Div2=3}}



После запуска, видно, что активной стала вычислительная ветвь, оканчивающая на ФУ D1\_1. Затормозим выполнение этой ветки, установив задержку выдачи операнда равным 100 для вершины M4\_1: Eventser.EventWaitSet={ TimeAtr=100 Mk={M4\_1.Mul1=pi}}. Вершина, для которой необходимо осуществить задержку определяется по потоковому графу. Т.к. имеется возможность задержки только исходных данных, необходимо определить, какое из исходных данных за необходимо для вычисления ветки, завершившейся первой. Для того, чтобы затормозить выполнение ветки, достаточно затормозить выдачу хотя бы одного исходного данного, необходимого для нее. Задержку можно осуществить с помощью атрибута TimeAtr, задающего время поступления данного в ВС. В данном случае, мы задерживаем выдачу данных, на 100 единиц модельного времени.



**Исследование вычислительной системы при приходе исходных данных в случайные моменты времени.**

В данном разделе исследования необходимо случайным образом задать время поступления всех исходных данных (время должно быть не слишком большим). Провести 2-3 запуска моделирования. Проанализировать полученные результаты. Например, можно для моделирования взять следующие времена задержки:

Eventser.EventWaitSet={ TimeAtr=1 Mk={M1\_1.Mul1=r1}}

Eventser.EventWaitSet={ TimeAtr=1 Mk={M1\_1.Mul2=r1}}

Eventser.EventWaitSet={ TimeAtr=2 Mk={M2\_1.Mul1=r1}}

Eventser.EventWaitSet={ TimeAtr=2 Mk={M2\_1.Mul2=r2}}

Eventser.EventWaitSet={ TimeAtr=3 Mk={M3\_1.Mul1=r1}}

Eventser.EventWaitSet={ TimeAtr=3 Mk={M3\_1.Mul2=r2}}

Eventser.EventWaitSet={ TimeAtr=4 Mk={M4\_1.Mul1=pi}}

Eventser.EventWaitSet={ TimeAtr=4 Mk={M4\_1.Mul2=h}}

Eventser.EventWaitSet={ TimeAtr=5 Mk={M5\_1.Mul1=x}}

Eventser.EventWaitSet={ TimeAtr=5 Mk={M5\_1.Mul2=s1}}

Eventser.EventWaitSet={ TimeAtr=6 Mk={M6\_1.Mul2=s2}}

Eventser.EventWaitSet={ TimeAtr=7 Mk={M7\_1.Mul1=s1}}

Eventser.EventWaitSet={ TimeAtr=7 Mk={M7\_1.Mul2=s2}}

Eventser.EventWaitSet={ TimeAtr=7 Mk={M7\_2.Mul1=h}}

Eventser.EventWaitSet={ TimeAtr=2 Mk={S2\_1.Sum1=s1}}

Eventser.EventWaitSet={ TimeAtr=2 Mk={S2\_2.Sum2=s2}}

Eventser.EventWaitSet={ TimeAtr=3 Mk={S3\_1.Sub1=x}}

Eventser.EventWaitSet={ TimeAtr=3 Mk={S3\_1.Sub2=h}}

Eventser.EventWaitSet={ TimeAtr=1 Mk={D1\_1.Div2=3}}

Eventser.EventWaitSet={ TimeAtr=2 Mk={D2\_1.Div2=3}}

Eventser.EventWaitSet={ TimeAtr=3 Mk={D3\_1.Div2=3}}

***Критерии оценки работы 1:***

4 балла: создана и запущена dataflow-программа

5 баллов: произведено моделирование вычислительного процесса

6 баллов: количество вершин в потоковом графе не менее 20

7 баллов: количество альтернативных ветвей в алгоритме не менее 3

8 баллов: имеется хорошо оформленный отчет о работе

9 баллов: количество вершин в потоковом графе не менее 30

10 баллов: количество вершин в потоковом графе не менее 40

**4.2 Практическая работа 2: Синтаксического разбор конструкции языка высокого уровня**

Целью работы является создание синтаксического анализатора конструкции языка высокого уровня на базе data-flow ВС ОА архитектуры. Реализация синтаксического анализатора производится в среде ОА программирования и моделирования.

**4.2.1 Варианты заданий для работы 2**

В качестве задания предлагается синтаксические конструкции различных языков программирования высокого уровня. Синтаксис описывается с помощью нотации расширенной формы Бекуса-Наура (РБНФ) [Ахо, Альфред В., Лам, Моника С., Сети, Ульман Джеффри Д. Компиляторы: Принципы, технологии, инструментарий, 2-е изд. Пер. с англ. - М.: ООО "И.Д. Вильямс", 2008 - 1184 с.]. Варианты приведены в таблице … .

Табл. … Варианты заданий для работы 2

|  |  |
| --- | --- |
|  | **Задание** |
| 1 | “for” I=D (“ to ” | “ downto ”) D “\_” I=D(“+”|“-”)D) “next” |
| 2 | I=(((I | DF) (“+” | “-“ | “/” | “\*”) (I | DF)) | S)  I=(((I | DF) (“+” | “-“ | “/” | “\*”) (I | DF)) | S)  … |
| 3 | “do while ” I (“<” | “>”) D I=(I | D) (“+” | “-“ | “/” | “\*”) (I | D) “loop” |
| 4 | “do until ” I ( “=” | “<>”) D “\_” I=(I | D) [(“+” | “-” | “/” | “\*”) (I | D)] “loop” |
| 5 | “do” I=(I | DZ) (“+” | “-“ | “/” | “\*”) (I | D) “loop while” I (“<”,”>”) (D,I) |
| 6 | “do” I=(I | D) [(“+” | “-“ | “/” | “\*”) (I | D)] “loop until” I (“=” | “<>”) (D|I) |
| 7 | “if ” I [(“=” | “<>” | “>” | “<”) (DZ | I)] “ then ” I=(I | DF) (“+” | “-”) (I | DF) [“else” I=(I | D) (“+” | “-”) (I | D)] “ endif” |
| 8 | “if ” I (“>” | “<”) (I | D) “ then” I=(I | D) [(“+” | “-”) (I | D)] [“ else ” I=(I | D) (“+” | “-”) (I | D)] “endif” |
| 9 | “if” I (“=” |”<>” | “>” | “<”) (DZ | I) “ and ” I [(“=” | “<>” | “>” | “<”) (DZ | I) “ then ” I=(I | DF) (“+” | “-” ) (I | DF)] “ endif” |
| 10 | “if ” I (“>” | “<”) (D | I) (“and” | “or”) I (“>” | “<”) (D,I) “then ” I=(I | D)[ (“+” | “-”) (I | D)] “endif” |
| 11 | “do while ” [ “not ” ] I (“<” | “>”) D “\_“ I=(I | D)[ (“+” | “-”)(I | D)] “ loop” |
| 12 | “do until” I (“=” | “<>”) D (“or” | “xor” | “and”) I (“=”,”<>”) D I=(I | D) (“+” | “-” | “/” | “\*”) I “ loop” |
| 13 | “dim” I [ “as” (“­integer” | “long” | “variant” | “string”)] |
| 14 | “dim" I”(“ (D | D,D) “)” (\_ | “as” (­“integer” | “long” | “string”)) |
| 15 | (I “=” B”(“I”)” | I = B”(“D”)” | I = B”(“I | “I”) | I = B”(“D | D”)” | I = B”(“D | I “)” | I = B”(“D | I”)”) |
| 16 | (I “=” B”(“I “,” I”)” | I = B”(“D “,” D “)” | I = B”(“D,I”)” | I = B”(“D,I”)”) (+ | - | / | \*) (I | D) |
| 17 | I=(I | DF) (“+”| “-“) (I | DF) (“+”| “-“) (I | DF)  I=(I | DF) (“+”| “-”) (I | DF) (“+”| “-”) (I | DF)  … |
| 18 | B “(” ( (I | D | DF) [(“+” | “-” | “/” | “\*”) (I | DF)] ) | (I | D) | S) “)”  B “(” ( (I | D | DF) [(“+” | “-” | “/” | “\*”) (I | DF)]) | (I | D) | S) “)”  … |
| 19 | B “(“ (I | DF) (+ | - | / | \*) (I | DF) | (I | D) “)”  B “(“ (I | DF) (+ | - | / | \*) (I | DF) | (I | D) “)”  … |
| 20 | I= B”(“ (I | D), ( (I | D) ) (“+”| “-”) B( (I | D) “)”  I= B”(“ (I | D), ( (I | D) ) (“+”| “-”) B( (I | D) “)”  … |
| 21 | I= B”(“ (I | D)[ (“+”| “-”) (I | D)] “,” ( ( (I | D) [(“+”| “-”) (I | D)]) | S) “)”  I= B”(“ (I | D) [(“+”| “-”) (I | D)] “,” ( ( (I | D) [(“+”| “-”) (I | D)]) | S) “)”  … |
| 22 | I= B”(“ (I | D) [, (I | D) ) ) (+ | - ) (I | D)]”)”  I= B”(“ (I | D) [, (I | D) ) ) (+ | - ) (I | D)]”)”  … |
| 23 | I=( (I | D) (“+” | “-“ | “/” | “\*”) (I | D) ) (“+” | “-“ | “/” | “\*”) ( (I | D) (“+” | “-” | “/” | \*) (I | D) ) |
| 24 | I=( (I | D) (“+” | “-” | “/” | “\*”) (I | D) ) (“+” | “-” | “/” | “\*”) ( (I | D) (“+” | “-” | “/” | “\*”) (I | D) ) |
| 25 | I = I (“=” | “<>” | “>” | “<”) I (“ and ” | “ or ”) I (“=” | “<>” | “>” | “<”) I  I = I (“=” | “<>” | “>” | “<”) I (“ and ” | “ or ”) I (“=” | “<>” | “>” | “<”) I  … |
| 26 | I = ((“true” | “false”) | (I (“=” | “<>” | “>” | “<”) I))  I = ((“true” | “false”) | (I (“=” | “<>” | “>” | “<”) I))  … |
| 27 | I = ( ( “true” | “false ” | I) | (I (“=” | “<>” | “>” | “<”) I))  I = ( ( “true” | “false ” I) | (I (“=” | “<>” | “>” | “<”) I))  … |
| 28 | I=(“true | “false” | I) (“and” | “or” | “xor”) I (“=” | “<>” | “>” | “<”) I  I=(“true | “false” | I) (“and” | “or” | “xor”) I (“=” | “<>” | “>” | “<”) I  … |
| 29 | B ((“true | “false” | I) | (I | D) , (“true” | false” | I) | (I | D)) |
| 30 | I “=” ((I (“=” | “<>” | “>” | “<”) (I | D)) | (I (“and” | “or” | “xor”) I))  I “=” (I (“=” | “<>” | “>” | “<”) (I | D)) | (I (“and” | “or” | “xor”) I))  … |
| 31 | I “=” ( («true» | «false») | ( (I | DF) («and» | «or» | «xor») I))…  I “=” ( («true» | «false») | ( (I | DF) («and» | «or» | «xor») I))…  … |
| 32 | I “=” ( ( (I | DF) (“+” | “-” | “/” | “\*”) (I | DF) ) | (I (“and” | “or” | xor”) I) )  I “=” ( ( (I | DF) (“+” | “-” | “/” | “\*”) (I | DF) ) | (I (“and” | “or” | “xor”) I) )  … |

**Обозначения:**

(Для задания выражений используется расширенная форма Бекуса-Наура.)

Alpha= “a”| “b” | “c” | … | “z” | “A” | “B” | … | “Z”

Figure= “0”| “1”| “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

I – имя переменной; I=Alpha (Alpha | Figure)\*;

D – целое десятичное число D= Figure (Figure)\*;

DF – дробное десятичное число DF=D “.” D;

DZ – десятичное число со знаком (“+” | “-”) D;

B – имя процедуры (функции) I=Alpha (Alpha | Figure)\*;

**S** – строковая константа, т.е. строка, заключенная в кавычки;

\_ - пробел (или пустой символ, т.е. на этом месте нет никакой лексемы)

| - альтернатива (может применяться либо одна, либо другая конструкции)

[ ] – языковая конструкция может не присутствовать или присутствовать только один раз.

( ) – группировка различных операций в выражении, описывающем синтаксис языка

\* - выражение повторяется от 0 до бесконечного количества раз.

**4.2.2 Составление графа состояний распознающего автомата**

Синтаксический анализатор работает на базе распознающего автомата [Ахо, Альфред В., Лам, Моника С., Сети, Ульман Джеффри Д. Компиляторы: Принципы, технологии, инструментарий, 2-е изд. Пер. с англ. - М.: ООО "И.Д. Вильямс", 2008 - 1184 с.], который впоследствии реализуется в data-flow парадигме.

Система синтаксического анализ на базе ОА представляет собой следующее [Салибекян С.М., Панфилов П.Б. Анализ языка с помощью объектно-атрибутного подхода к организации вычислений // Программная инженерия. №1, 2013 – стр. 9-16].

Проиллюстрируем создание графа состояний автомата на примере анализа следующей синтаксической конструкции языка высокого уровня, представленной в РБНФ:

I “=” ( ( (I | DF) (“+” | “-” | “/” | “\*”) (I | DF) ) | (I (“and” | “or” | xor”) I) ) (…)

I “=” ( ( (I | DF) (“+” | “-” | “/” | “\*”) (I | DF) ) | (I (“and” | “or” | “xor”) I) )

…

По РБНФ можно построить граф состояний распознающего автомата [Ахо, Альфред В., Лам, Моника С., Сети, Рави, Ульман, Джеффри Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд.: Пер. с англ. – М.: ООО «ИД Вильямс», 2008. 1184 с.]. Каждое состояние автомат следует поименовать, т.к. в ОА компиляторе каждому такому состоянию будет соответствовать свое ФУ. Имена состояний в ОА-программе будут являться именами соответствующих ФУ. Для (…) можно составить такой граф состояний распознающего автомата (рис. …). Пометкой «Id» обозначается идентификатор.



Рис. … Граф состояний распознающего автомата

**4.2.3 Создание ОА программы синтаксического анализатора**

Для реализации синтаксического анализатора необходимо применение ФУ типов: FULex – лексический разбор текста, FUList – список ИК. Милликоманды для этих типов ФУ приведены в приложении Г.

В начале файла должен присутствовать заголовок для настройки программы для работы в новой версии среды ОА-программирования. Этот заголовок неизменный и его необходимо только перенести в создаваемый файл АО-программы:

\\ Для создания индексного файла сначала надо запустить программу !!!!

CapsManager.IndexVectCreate=20000

CapsManager.IpIcIdOutMk=MainBus.IpIcIdSet

CapsManager.IpIcIdOutMk=VariableManager.IpIcIdSet

CapsManager.IpIcIdOutMk=ListSyntez.IpIcIdSet

MainBus.ModeSet=1

\\ -------------------------------------- \\

MainBus.FUTypeCorrectSet=-96 // Корректировка номера типа ФУ для переноса на новую ОА-платформу

Далее идет секция программы с описанием ФУ, входящих в ОА-систему. В системе обязательно должны присутствовать два ФУ: консоль и лексический анализатор. Их объявления выглядят так:

NewFU={Mnemo="Lex" FUType=FULexNew}

NewFU={Mnemo="Console" FUType=FUConsNew}

После этого в программе присутствуют описания ФУ, ассоциированных с состояниями распознающего автомата. Все эти ФУ имеют тип FUList («Список»). Их количество соответствует количеству состояний автомата их имена идентичны именам состояний графа состояний распознающего автомата. Например, для графа на рис. … объявления ФУ-состояний будет следующим:

NewFU={Mnemo="Start" FUType=FUListNew Hint="Стартовое состояние"}

NewFU={Mnemo="After\_Id" FUType=FUListNew}

NewFU={Mnemo="AfterEqv" FUType=FUListNew}

NewFU={Mnemo="AfterDF" FUType=FUListNew}

NewFU={Mnemo="Logic" FUType=FUListNew}

NewFU={Mnemo="Arithm" FUType=FUListNew} После описания ФУ-состояний необходимо поместить еще две строки кода для настройки ОА-среды:

MainBus.PartialResetSet \\ Запомнить число созданных ФУ

MainBus.ModeSet=2

Далее идет секция описания синтаксического анализа. Начинается она с установки разделителей для ФУ лексического анализа. Разделители могут быть как простыми (состоят из одного символа), так и сложными (состоят из нескольких символов, например: «==», «++», «<=>»). Установка осуществляется с помощью МК SepAdd, а в нагрузке указывается строка, что будет считаться разделителем в тексте компилируемой программы. Для установки необходимых разделителей в нашем примере необходимо выполнить МК:

Lex.SepAdd="="="+"="-"="\*"="/"

Нам также потребуется задать для ФУ лексического разбора мнемоники служебных слов «ложь» и «правда». Сделаем это с помощью двух МК:

Lex.FalseAdd="false"

Lex.TrueAdd="true"

Также можно настроить реакцию программы на лексическую ошибку. Это делается посредством МК Lex.ErrProgSet. В нагрузке данной МК находится ИК с программной, запускаемой при возникновении ошибки. Например, можноо описать реакцию компилятора на лексическую ошибку так:

Lex.ErrProgSet={Console.OutLn="Wrong lexema"}

Следующая секция – настройку атрибутов лексем. В ней задаются мнемоники атрибутов для типов лексем. Данные настройки лучше перекопировать в программу без изменений:

Sep\*-4

Int\*-10

Float\*-3

Bool\*-5

Str\*-6

Следующая МК необходима для настройки начального состояния распознающего автомата.

Lex.ReceiverMkSet=Start.FindAnd // Настройка на начальное состояние

МК Lex.ReceiverMkSet дает указание ФУ лексического разбор, какой атрибут необходимо прикреплять в очередной выделенной лексеме. В данном случае к первой выделенной лексеме прикрепляется атрибут Start.FindAnd, и сформированная МК выдается на Шину. Шина же переправляет МК на ФУ «Start». Далее необходимо описать все состояний распознающего автомата.

Следующая секция программы – настройка ФУ-состояний. Произведем такую настройку на примере состояния AfterId. Как видно, из данного состояния выходят две дуги, следовательно, в списке будет три строки: две для описания перехода и одна для обработки ошибки компиляции. Настройка данного ФУ приведена ниже:

After\_Id\_2.Set=

>{Mnemo="and"="or"="xor" Lex.ReceiverMkSet=Logic.FindAnd}

>{Sep="+"="-"="\*"="/" Lex.ReceiverMkSet=Arithm.FindAnd}

>{0 Lex.Stop Console.OutLn="No correct operation description"}

Первая строка задает переход в состояние Logic, вторая – в состояние Arithm. В третьей строке выводится сообщение о синтаксической ошибке. “0” в начале ИК обозначает универсальный атрибут, т.е. атрибут, который совпадает с любой ИК; таким образом, если в первых двух строках совпадения не обнаружится, то третья строка выполнится обязательно. А с помощью МК Lex.Stop производится остановка лексического анализатор, т.к. не должен продолжать свою работу после обнаружения ошибки в тексте программы. Другие состояния программируются аналогично.

***Внимание***. Для отладки программы компилятора можно применять вывод различных меток внутри программы. Например, для того, чтобы проверить, произошел ли переход из состояния AftrId в состояние Logic, можно скорректировать программу (…) так:

AfterId.Set=

>{Mnemo="and"="or"="xor" Lex.ReceiverMkSet=Logic.FindAnd

Console.OutLn=”From AfterId to Logic”}

>{Sep="+"="-"="\*"="/" Lex.ReceiverMkSet=Arithm.FindAnd}

>{0 Lex.Stop Console.OutLn="No correct operation description"}

***Внимание***. Если для описания переход важен только атрибут (тип) лексему, то в этом случае записывается только атрибут ИП, а нагрузка не указывается. Например, описание состояние Start, будет выглядеть следующим образом:

А затем после успешной отладки МК Console.OutLn удаляется из ОА-программы.

В самом конце ОА программы помещается МК начала лексического анализа («Lex.Lexing»), а в нагрузке – анализируемая строка.

Затем необходимо проверить, находится ли распознающий автомат после проведения синтаксического анализа в конечном состоянии. Такая проверка осуществляется по МК для приемника лексем. Задать МК для финальных состояний можно с помощью МК FinMkAdd. Для нашего примере состояние Start является как начальным, так и конечным. Поэтому назначим для лексера МК для конечного состояния с помощью МК Lex.FinMkAdd=Start.FindAnd. А МК Lex.FinMkProgExec={Console.OutLn="Correct"} запускает программу реакции на случай нахождения распознающего автомата в конечном состоянии. МК Lex.NoFinMkProgExec={Console.OutLn="Wrong sintacsis"}, соответственно на случай ненахождении автомата в конечном состоянии.

Полный текст ОА-программы синтаксического анализатора приведен в приложении Д.

**4.2.4 Порядок запуска лексического анализатора**

Программа лексического анализатора пишется и компилируется в старой версии среды ОА программирования и моделирования, а запускается в новой. Для переноса программы на новую платформу старая версия среды создает так называемый индексный файл, содержащий описание программы во внутреннем представлении (аналогично байт-коду JAVA-машины (JVM) [Meyer J., Downing T. Java Virtual Machine. – O'Reilly Media, 1997. – С. 399–400.]). Для создания индексного файла необходимо сделать следующие действия:

- Запустить ОА-программу

- выбрать пункт меню «Файл->Создать индексный файл» и записать индексный файл в директорию новой версии среды ОА-программирования и моделирования.

- в консоли перейти в директорию новой версии среды (MillicomNew) и в командной строке вызвать millicom название индексного файла.

Внимание, при корректировке ОА-программы для ее запуска необходимо проделывать все вышеописанные действия заново.

***Литература***

1. Берж К. Теория графов и ее применения. М., изд. Иностранной литературы. 1962.
2. Уилсон Р. Введение в теорию графов. М.: Мир, 1977.
3. Корнеев В.В., Киселев А.В. Современные микропроцессоры. – 3-е издание, перераб. и доп. – СПб.: БХВ-Петербург. 2003. ­– 448 с.
4. Компьютеры на СБИС: В 2-х кн. Кн.1/Пер. с япон. Мотоока Т., Томита С., Танака Х. и др. – Мир, 1988. – 392 с.
5. Велях Е. Последовательно-параллельные вычисления: Пер. с англ. - М.: Мир, 1985.- 456 с.
6. Введение в алгоритмы параллельных вычислений /Молчанов И.Н.; Отв. ред. Яковлев М.Ф.; АН УССР. Ин-т кибернетики им. В.М. Глушкова. – Киев: Наук. думка, 1990. – 128 с.
7. Параллельные вычисления /Под ред. Г. Родринга: Пер. с англ. /Под ред. Ю.Г. Дадаева. – М.: Наука. Гл. ред. физ.-мат. лит., 1986. – 376 с.
8. Салибекян С.М., Панфилов П.Б. Объектно-атрибутная архитектура – новый подход к созданию объектных систем // Информационные технологии. 2012, №2 стр. 8-14

**Приложение А. Список милликоманд ФУ «Потоковое АЛУ с плавающей точкой» и ФУ «консольный вывод»**

**Общие милликоманды**

**OperandsReset**: Сброс операндов

**ReceiverMkSet**:Установить милликоманду для потребителя (автоматически создается новая запись описания потребителя

**ReceiversReset**: Сбросить список потребителей

**FireingProgSet**: Установить указатель на подпрограмму, запускаемую при активизации вершины (получении результата выполнения операции)"

**PopMk**: "Выдать результат выполнения операции"

**AngleModeSet**: Установить режим измерения угла (0- радиены, 1 - градусы).

**AutoDisactiveModeSet**: Установить режим автоматического перехода ФУ в пассивный режим после получения выходных данных (нужно для того, чтобы предотвратить нежелательную повторную активацию ФУ, которая может нарушить вычислительный процесс).

**ActiveSet**: Установить режим активности (true на входе – активный режим, false – пассивный).

**FiringProgSet** – установить программу, запускаемую при приходе полного комплекта данных к ФУ. Программа оформляется в виде ИК. Например, «Dev1.FiringProgSet={Cons.Out=”Thread1:” Dev1.PopMk=Cons.OutLn}». Данный фрагмент кода выводит надпись ”Thread1:”, чтобы обозначить номер ветви вычисления, по которой получен результат, а затем выводит результат вычисления.

**Арифметические функции**

**NOperandsSet**: Установить число операндов для операций с накоплением

**Sum**: Сложение с накоплением.

**Sub**: Вычитание с накоплением.

**Mul**: Умножение с накоплением.

**Div**: Деление с накоплением.

**Sum1**: Первое слагаемое.

**Sum2**: Второе слагаемое.

**Sub1**: Вычитаемое.

**Sub2**: Вычитатель.

**Mul1**: Умножаемое.

**Mul2**: Умножитель.

**Div1**: Делимое.

**Div2**: Делитель.

**Sqr**: Квадрат.

**Sqrt**: Квадратный корень.

**Тригонометрические операции**

**Sin**: Синус.

**Cos**: Косинус.

**Tg**: Тангенс.

**Ctg**: Котангенс.

**Asin**: Арксинус.

**Acos**: Арккосинус.

**Atg**": Арктангенс.

**Настройки моделирования**

**RegLoadTimeSet**: Установить время записи во внутренний регистр.

**SumTimeSet**: Установить время суммирования.

**SubTimeSet**: Установить время вычитания.

**MulTimeSet**: Установить время умножения.

**DivTimeSet**: Установить время деления.

**SqrTimeSet**: Установить время вычисления квадрата.

**SqrtTimeSet**: Установить время вычисления квадратного корня.

**SinTimeSet**: Установить время вычисления синуса.

**CosTimeSet**: Установить время вычисления косинуса.

**TgTimeSet**: Установить время вычисления тангенса.

**CtgTimeSet**: Установить время вычисления котангенса.

**AsinTimeSet**: Установить время вычисления арксинуса.

**AcosTimeSet**: Установить время вычисления арккосинуса.

**AtgTimeSet**: Установить время вычисления квадратного арктангенса.

**Милликоманды консоли вывода:**

**Console.LnOut**: вывод константы или переменной на консоль с новой строки.

**Console.Out**: вывод константы или переменной на консоль на той же строке.

**Console.OutLn**: вывод константы или переменной на консоль и переход к новой строке.

**Приложение Б. Миллипрограмма расчета объёма усеченного конуса**

//M-Mult

NewFU={Mnemo="M1\_1" FUType=FUStreamFloatAlu}

NewFU={Mnemo="M2\_1" FUType=FUStreamFloatAlu}

NewFU={Mnemo="M3\_1" FUType=FUStreamFloatAlu}

NewFU={Mnemo="M4\_1" FUType=FUStreamFloatAlu}

NewFU={Mnemo="M4\_2" FUType=FUStreamFloatAlu}

NewFU={Mnemo="M5\_1" FUType=FUStreamFloatAlu}

NewFU={Mnemo="M6\_1" FUType=FUStreamFloatAlu}

NewFU={Mnemo="M7\_1" FUType=FUStreamFloatAlu}

NewFU={Mnemo="M7\_2" FUType=FUStreamFloatAlu}

//S-Sum

NewFU={Mnemo="S1\_1" FUType=FUStreamFloatAlu}

NewFU={Mnemo="S1\_2" FUType=FUStreamFloatAlu}

NewFU={Mnemo="S2\_1" FUType=FUStreamFloatAlu}

NewFU={Mnemo="S2\_2" FUType=FUStreamFloatAlu}

NewFU={Mnemo="S3\_1" FUType=FUStreamFloatAlu}

NewFU={Mnemo="S3\_2" FUType=FUStreamFloatAlu}

//D-Div

NewFU={Mnemo="D1\_1" FUType=FUStreamFloatAlu}

NewFU={Mnemo="D2\_1" FUType=FUStreamFloatAlu}

NewFU={Mnemo="D3\_1" FUType=FUStreamFloatAlu}

//R-Root

NewFU={Mnemo="R1\_1" FUType=FUStreamFloatAlu}

//Init

r1#3

r2#2

x#6

h#2

s1#28.2743338

s2#12.5663706

//line 2

S3\_1.ReceiverMkSet=M6\_1.Mul1

M5\_1.ReceiverMkSet=S3\_2.Sub1

M1\_1.ReceiverMkSet=S1\_1.Sum1

M3\_1.ReceiverMkSet=S1\_1.Sum2

M2\_1.ReceiverMkSet=S1\_2.Sum2

M4\_1.ReceiverMkSet=M4\_2.Mul2

M7\_1.ReceiverMkSet=R1\_1.Sqrt

//line 3

S1\_1.ReceiverMkSet=S1\_2.Sum1

R1\_1.ReceiverMkSet=S2\_1.Sum2

M6\_1.ReceiverMkSet=S3\_2.Sub2

//line 4

S1\_2.ReceiverMkSet=M4\_2.Mul1

S2\_1.ReceiverMkSet=S2\_2.Sum1

S3\_2.ReceiverMkSet=D3\_1.Div1

D3\_1.ReceiverMkSet=Console.LnOut

//line 5

M4\_2.ReceiverMkSet=D1\_1.Div1

D1\_1.ReceiverMkSet=Console.LnOut

S2\_2.ReceiverMkSet=M7\_2.Mul2

//line 6

M7\_2.ReceiverMkSet=D2\_1.Div1

D2\_1.ReceiverMkSet=Console.LnOut

//main line

M1\_1.Mul1=r1

M1\_1.Mul2=r1

M2\_1.Mul1=r2

M2\_1.Mul2=r2

M3\_1.Mul1=r1

M3\_1.Mul2=r2

M4\_1.Mul1=pi

M4\_1.Mul2=h

M5\_1.Mul1=x

M5\_1.Mul2=s1

M6\_1.Mul2=s2

M7\_1.Mul1=s1

M7\_1.Mul2=s2

M7\_2.Mul1=h

S2\_1.Sum1=s1

S2\_2.Sum2=s2

S3\_1.Sub1=x

S3\_1.Sub2=h

D1\_1.Div2=3

D2\_1.Div2=3

D3\_1.Div2=3

**Приложение В. Миллипрограмма с настройками для моделирования**

//Modelling

NewFU={Mnemo="Gant" FUType=FUGant}

NewFU={Mnemo="Eventser" FUType=FUEventser}

NewFU={Mnemo="Scheduler" FUType=FUScheduler}

NewFU={Mnemo="Mul1" FUType=FUStreamFloatAlu}

Eventser.CurrentTimePointPopMk=Gant.CurrentTimeRefSet

Eventser.ContextPopMk=Scheduler.EventserContextSet

Scheduler.ContextPopMk=Mul1.SchedulerContextSet

Eventser.OutProgSet={Eventser.EventCapsPopMk=Gant.EventSet}

Eventser.EventRequestProgSet={Eventser.EventRequestCapsPopMk=Gant.OperandsSet}

Eventser.FinProgSet={Gant.Draw Console.LnOut="Время выполнения программы: " Eventser.CurrentTimePointPopMk=Console.Out

Console.LnOut="Коэффициент параллелиизма: " Eventser.ParallelFactorPopMk=Console.LnOut}

Scheduler.NCoresSet=4

Mul1.SubTimeSet=2

Mul1.MulTimeSet=9

Mul1.DivTimeSet=10

Mul1.SumTimeSet=2

Mul1.SqrtTimeSet=8

Mul1.RegLoadTimeSet=0.2

ContextTemplate=nil

Mul1.ManualModeSet=true

Mul1.ContextPop=ContextTemplate

//M-Mult

NewFU={Mnemo="M1\_1" FUType=ContextTemplate}

NewFU={Mnemo="M2\_1" FUType=ContextTemplate}

NewFU={Mnemo="M3\_1" FUType=ContextTemplate}

NewFU={Mnemo="M4\_1" FUType=ContextTemplate}

NewFU={Mnemo="M4\_2" FUType=ContextTemplate}

NewFU={Mnemo="M5\_1" FUType=ContextTemplate}

NewFU={Mnemo="M6\_1" FUType=ContextTemplate}

NewFU={Mnemo="M7\_1" FUType=ContextTemplate}

NewFU={Mnemo="M7\_2" FUType=ContextTemplate}

//S-Sum

NewFU={Mnemo="S1\_1" FUType=ContextTemplate}

NewFU={Mnemo="S1\_2" FUType=ContextTemplate}

NewFU={Mnemo="S2\_1" FUType=ContextTemplate}

NewFU={Mnemo="S2\_2" FUType=ContextTemplate}

NewFU={Mnemo="S3\_1" FUType=ContextTemplate}

NewFU={Mnemo="S3\_2" FUType=ContextTemplate}

//D-Div

NewFU={Mnemo="D1\_1" FUType=ContextTemplate}

NewFU={Mnemo="D2\_1" FUType=ContextTemplate}

NewFU={Mnemo="D3\_1" FUType=ContextTemplate}

//R-Root

NewFU={Mnemo="R1\_1" FUType=ContextTemplate}

//Init

r1#3

r2#2

x#6

h#2

s1#28.2743338

s2#12.5663706

//line 2

S3\_1.ReceiverMkSet=M6\_1.Mul1

M5\_1.ReceiverMkSet=S3\_2.Sub1

M1\_1.ReceiverMkSet=S1\_1.Sum1

M3\_1.ReceiverMkSet=S1\_1.Sum2

M2\_1.ReceiverMkSet=S1\_2.Sum2

M4\_1.ReceiverMkSet=M4\_2.Mul2

M7\_1.ReceiverMkSet=R1\_1.Sqrt

//line 3

S1\_1.ReceiverMkSet=S1\_2.Sum1

R1\_1.ReceiverMkSet=S2\_1.Sum2

M6\_1.ReceiverMkSet=S3\_2.Sub2

//line 4

S1\_2.ReceiverMkSet=M4\_2.Mul1

S2\_1.ReceiverMkSet=S2\_2.Sum1

S3\_2.ReceiverMkSet=D3\_1.Div1

D3\_1.ReceiverMkSet=Console.LnOut

//line 5

M4\_2.ReceiverMkSet=D1\_1.Div1

D1\_1.ReceiverMkSet=Console.LnOut

S2\_2.ReceiverMkSet=M7\_2.Mul2

//line 6

M7\_2.ReceiverMkSet=D2\_1.Div1

D2\_1.ReceiverMkSet=Console.LnOut

//main line

Eventser.EventWaitSet={ TimeAtr=0 Mk={M1\_1.Mul1=r1}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={M1\_1.Mul2=r1}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={M2\_1.Mul1=r1}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={M2\_1.Mul2=r2}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={M3\_1.Mul1=r1}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={M3\_1.Mul2=r2}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={M4\_1.Mul1=pi}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={M4\_1.Mul2=h}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={M5\_1.Mul1=x}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={M5\_1.Mul2=s1}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={M6\_1.Mul2=s2}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={M7\_1.Mul1=s1}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={M7\_1.Mul2=s2}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={M7\_2.Mul1=h}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={S2\_1.Sum1=s1}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={S2\_2.Sum2=s2}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={S3\_1.Sub1=x}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={S3\_1.Sub2=h}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={D1\_1.Div2=3}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={D2\_1.Div2=3}}

Eventser.EventWaitSet={ TimeAtr=0 Mk={D3\_1.Div2=3}}

Eventser.Start

**Приложение Г. Список милликоманд ФУ «Синтаксический анализатор» (FULex) и «Список» (FUList)**

**«Список» (FUList)**

- **Set**: Установить ссылку на список

- **Out**: Выдать указатель на начало списка

- **OutMk**: Выдать милликоманду с указателем на начало списка

- **FindAnd**: Принять запись для сравнения

**«Синтаксический анализатор» (FULex)**

- ReceiverMkSet: Установить МК для приемника лексем

- SepAdd: Добавить разделить

- SepErase: Удалить разделитель из списка разделителей

- SepClear: Очистить список разделителей

- FalseAdd: Добавить мнемонику лжи

- TrueAdd: Добавить мнемонику правды

- TrueFalseClear: Очистить списки мнемоник правды и лжи

- FinMkAdd: Добавить финальную МК

- FinMkClear: Очистить список фитальных МК

- FinMkProgExec: Выполнить программу, если лексер в финальном состоянии

- NoFinMkProgExec: Выполнить программу, если лексер не в финальном состоянии

- Stop: Остановить лексический анализ

- Lexing: Лексический разбор строки

**Приложение Д. АО-программа синтаксического анализатора**

\\ Для создания индексного файла сначала надо запустить программу !!!!

CapsManager.IndexVectCreate=20000

CapsManager.IpIcIdOutMk=MainBus.IpIcIdSet

CapsManager.IpIcIdOutMk=VariableManager.IpIcIdSet

CapsManager.IpIcIdOutMk=ListSyntez.IpIcIdSet

MainBus.ModeSet=1

MainBus.FUTypeCorrectSet=-96 // Корректировка номера типа ФУ для переноса на новую ОА-платформу

\\ -------------------------------------- \\

NewFU={Mnemo="Lex" FUType=FULexNew}

NewFU={Mnemo="Console" FUType=FUConsNew}

NewFU={Mnemo="Start" FUType=FUListNew Hint="Стартовое состояние"}

NewFU={Mnemo="After\_Id" FUType=FUListNew}

NewFU={Mnemo="AfterEqv" FUType=FUListNew}

NewFU={Mnemo="After\_Id\_2" FUType=FUListNew}

NewFU={Mnemo="AfterDF" FUType=FUListNew}

NewFU={Mnemo="Logic" FUType=FUListNew}

NewFU={Mnemo="Arithm" FUType=FUListNew}

MainBus.PartialResetSet \\ Запомнить число созданных ФУ

MainBus.ModeSet=2

Lex.SepAdd="="="+"="-"="\*"="/"

Lex.FalseAdd="false"

Lex.TrueAdd="true"

Lex.ErrProgSet={Console.OutLn="Wrong lexema"}

//Mnemo!

Sep\*-4

Int\*-10

Float\*-3

Bool\*-5

Str\*-6

//Mnemo\*-2

Lex.ReceiverMkSet=Start.FindAnd // Настройка на начальное состояние

Start.Set=

>{Mnemo Lex.ReceiverMkSet=After\_Id.FindAnd}

>{0 Lex.Stop Console.OutLn="Identificator is not found"}

After\_Id.Set=

>{Sep="=" Lex.ReceiverMkSet=AfterEqv.FindAnd}

>{0 Lex.Stop Console.OutLn="Eqvition is not found"}

AfterEqv.Set=

>{Mnemo Lex.ReceiverMkSet=After\_Id\_2.FindAnd}

>{Float Lex.ReceiverMkSet=AfterDF.FindAnd}

>{0 Lex.Stop Console.OutLn="Wrong operator description"}

After\_Id\_2.Set=

>{Mnemo="and"="or"="xor" Lex.ReceiverMkSet=Logic.FindAnd}

>{Sep="+"="-"="\*"="/" Lex.ReceiverMkSet=Arithm.FindAnd}

>{0 Lex.Stop Console.OutLn="No correct operation description"}

AfterDF.Set=

>{Sep="+"="-"="\*"="/" Lex.ReceiverMkSet=Arithm.FindAnd}

>{0 Lex.Stop Console.OutLn="No correct operation description"}

Logic.Set=

>{Mnemo Lex.ReceiverMkSet=Start.FindAnd}

>{0 Lex.Stop Console.OutLn="No identificator"}

Arithm.Set=

>{Mnemo Float Lex.ReceiverMkSet=Start.FindAnd}

>{0 Lex.Stop Console.OutLn="No identificator"}

Lex.Lexing="X=y+z"

// Проверка конецного состояния распознающего автомата

Lex.FinMkAdd=Start.FindAnd

Lex.FinMkProgExec={Console.OutLn="Correct"}

Lex.NoFinMkProgExec={Console.OutLn="Wrong sintacsis"}

Карп Миллер – основы data-flow

Акторное программирование

Переделать рис. Планировщик-коммутатор

Доделать рис. Автомат-транслятор с магазинной памятью

Jerry Banks, John S. Carson II, Barry L Nelson, David M. Nicol Discrete-Event System Simulation – Найти точное библиографическое описание