Федеральное государственное бюджетное учреждение
Национальный исследовательский центр «Курчатовский институт»

Федеральное государственное учреждение «Федеральный научный центр
Научно-исследовательский институт системных исследований
Российской академии наук»
(ФГУ ФНЦ НИИСИ РАН)

ТРУДЫ НИИСИ РАН

ТОМ 13 № 1-2

МАТЕМАТИЧЕСКОЕ И КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ СЛОЖНЫХ СИСТЕМ:

ТЕОРЕТИЧЕСКИЕ И ПРИКЛАДНЫЕ АСПЕКТЫ

МОСКВА
2023

Тематика номера:
Вопросы программирования, математическое моделирование и визуализация систем виртуального окружения, многопроцессорные системы реального времени, математическое моделирование автономных агентов, информационные технологии в учебной информатике

Журнал публикует оригинальные статьи по следующим областям исследований: математика, математическое и компьютерное моделирование, обработка изображений, визуализация, системный анализ, методы обработки сигналов, информационная безопасность, информационные технологии, высокопроизводительные вычисления, оптико-нейронные технологии, микро- и наноэлектроника, математические исследования и вопросы численного анализа, история науки и техники.

The topic of the issue:

Programming issues, mathematical modeling and visualization of virtual environment systems, real time multiprocessor systems, mathematical modeling of autonomous agents, information technologies in educational informatics

The Journal publishes novel articles on the following research arias: mathematics, mathematical and computer modeling, image processing, visualization, system analysis, signal processing, information security, information technologies, high-performance computing, optical-neural technologies, micro- and nanoelectronics, mathematical researches and problems of numerical analysis, history of science and of technique.

Заведующий редакцией: В.Е.Текунов

Издатель: ФГУ ФНЦ НИИСИ РАН,
117218, Москва, Нахимовский проспект 36, к. 1

СОДЕРЖАНИЕ

CONTENT

Визуальная отладка с помощью gdbgui

В.А. Галатенко[1], К.А. Костюхин[2], А.А. Фролова[3]

[1]Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук», Москва, РФ, galat@niisi.ras.ru;

[2]Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук», Москва, РФ, kost@niisi.ras.ru;

[3]Федеральное государственное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук», Москва, РФ, frolova-a@niisi.ras.ru

Аннотация. Интерактивный отладчик gdb является основным средством отладки программ, разрабатываемых на инструментальной платформе Linux. Предоставляя широкие возможности для отладки, gdb имеет один (по мнению достаточно большого числа разработчиков) существенный недостаток – интерфейс командной строки. Многие разработчики по-прежнему предпочитают более удобные графические интерфейсы. В этой статье будет рассмотрена перспективная графическая оболочка для gdb, являющаяся браузерным клиентом, за счет чего у пользователей появляется возможность платформонезависимой отладки.

Ключевые слова: отладка, gdb, графическая оболочка, GUI

1. Введение

Отладчик gdb [1], предоставляющий пользователю интерфейс командной строки, используется при разработке программ в UNIX-подобных системах уже давно. Все это время для него создавалось и создается большое количество графических оболочек (так называемых фронтендов), взаимодействующих с gdb через специальный «машинный» интерфейс (gdb/MI [2]).

В этой статье речь пойдет о сравнительно новом интерфейсе отладчика gdb – веб-интерфейсе под названием gdbgui [3]. Само слово веб-интерфейс говорит о том, что для отладки можно использовать любой современный браузер, поддерживающий в данном случае typescript. На рис. 1 представлен внешний вид gdbgui.

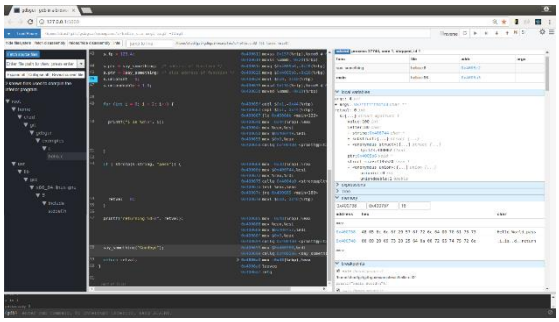


Рис. 1. Внешний вид gdbgui

2. Архитектура gdbgui

Технически gdbgui представляет собой веб-сервер, разворачиваемый на инструментальной ЭВМ, где располагается отладчик gdb. По умолчанию он использует порт 5000.

При установке соединения gdbgui запускает отладчик (по умолчанию командой gdb) и затем начинает сеанс отладки с использованием текстового интерфейса gdb/MI. По умолчанию (см. ниже) gdbgui также запускает локальный веб-браузер, с помощью которого предполагается вести отладку.

3. Установка gdbgui

Коммуникационная часть gdbgui написана на Python 3, в то время, как команды отладки – на typescript. Это удобно, поскольку дает возможность использовать gdbgui в разных операционных системах. В рамках данной статьи мы ограничимся описанием установки и функционирования gdbgui под управлением ОС семейства Linux.

Для установки gdbgui рекомендуется использовать команду pipx (на момент запуска этих команд python3 уже должен быть установлен в системе).

1. Установка pipx

python3 -m pip install --user pipx

2. Добавление нового каталога с выполняемыми программами в окружение пользователя

python3 -m userpath append ~/.local/bin

3. Установка gdbgui

pipx install gdbgui

Отметим, что gdbgui можно запускать и без установки командой

pipx run gdbgui

4. Отладка с помощью gdbgui

4.1. Параметры запуска gdbgui

Список всех параметров запуска можно посмотреть при помощи команды gdbgui –help.

```
$ gdbgui --help
usage: gdbgui [-h] [-g gdb_CMD] [-p PORT]
[--host HOST] [-r] [--auth-file AUTH_FILE]
[--user USER] [--password PASSWORD]
[--key KEY] [--cert CERT]
[--remap-sources REMAP_SOURCES]
[--project PROJECT] [-v] [-n] [-b BROWSER]
[--debug] [--args ...] [debug_program]
```

Ниже приводится их краткое описание.

debug_program - Отлаживаемый исполняемый файл и (опционально) его параметры командной строки. Чтобы передать параметры в отлаживаемый файл, их необходимо заключить в одинарные кавычки или использовать параметр --args.

Пример: gdbgui ./mybinary [другие параметры gdbgui]

gdbgui './mybinary –arg1 –arg2' [другие параметры gdbgui]

-g <команда для запуска gdb>, --gdb-cmd <команда для запуска gdb>

<команда для запуска gdb> может содержать параметры, с которыми отладчик должен быть запущен, например,
'/path/to/gdb --command=FILE -ix'.
По умолчанию запускается просто gdb.

-p <номер порта>, --port <номер порта>

Порт, через который gdbgui будет работать с удаленным пользователем. По умолчанию 5000.

--host <ip адрес или имя хоста>

Адрес или имя хоста, на котором будет работать gdbgui. По умолчанию 127.0.0.1.

-r, --remote

Ключ, указывающий, что ожидается удаленное соединение с gdbgui с другого хоста, поэтому локальный веб-браузер не запускается (отменить запуск локального веб-браузера можно и с помощью ключа -n).

--auth-file <файл с учетными данными>

Если указан этот ключ, то после установки соединения с веб-браузером, будет запрошена учетная информация (логин, пароль). Учетные данные для сверки (логин и пароль) должны находиться в указанном файле в текстовом виде, разделенные переводом строки.

--cert <SSL сертификат>

Включает авторизацию по SSL. В качестве аргумента указывается SSL сертификат. Сгенерировать сертификат можно следующей командой:

```
openssl req –newkey rsa:2048 -nodes -keyout
host.key -x509 -days 365 –out host.cert
```

--project <Каталог исходных текстов проекта>

При старте gdbgui строит дерево подкаталогов исходных текстов проекта из каталога, переданного в качестве аргумента.

-v, --version

Печатает текущую версию gdbgui.

-b <веб-браузер>, --browser <веб-браузер>

Локально запускает указанный веб-браузер вместо прописанного в системе по умолчанию.

--debug

Запуск gdbgui в режиме отладки. Если указан этот ключ, то в отдельном окне будут отображаться вызовы и результаты их выполнения gdb/MI интерфейса.

4.2. Обзор графического интерфейса

Проиллюстрируем функционал программы gdbgui на примере с рисунка 1.

В верхней части страницы находится поле «*Load Binary*», которое можно использовать для загрузки исполняемых файлов программы, а также передавать им аргументы, как в командной строке (рис. 2).

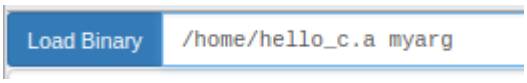


Рис. 2. Поле для загрузки файлов gdbgui

Там же справа располагаются кнопки управления ходом выполнения отлаживаемой программы: запустить/перезапустить, продолжить/остановить, продолжить по шагам (по строкам исходного кода) с заходом в вызываемую функцию или нет, выполнить одну команду процессора с заходом в вызываемую функцию или нет (рис. 3).

Рис. 3. Кнопки управления gdbgui

Для некоторых из этих элементов управления также есть сочетание клавиш.
Запуск программы: r
Продолжение выполнения: c
Шаг без захода в функцию: n или стрелка вправо.
Шаг с заходом в функцию: s или стрелка вниз.

В нижней части страницы находится традиционный интерфейс командной строки gdb (рис. 4).

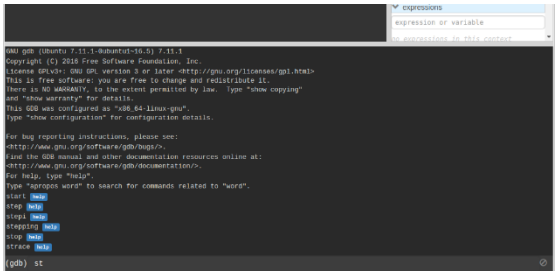


Рис. 4. Консоль gdb

Им можно воспользоваться для ввода команд отладчика, не имеющих своего отдельного графического элемента управления. Например, задать специфическую цель отладки командой target.

После загрузки исполняемого файла становится доступен раздел в середине страницы, в котором отображается исходный код программы (рис. 5) с расставленными точками останова.

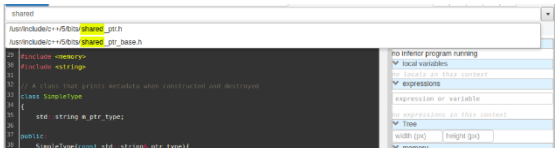


Рис. 5. Исходный текст программы

При необходимости можно здесь же смотреть дизассемблированный код программы (рис. 6).

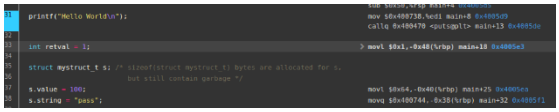


Рис. 6. Исходный и дизассемблированный текст

Если исходный текст недоступен, то отображается только дизассемблированный код (рис. 7).

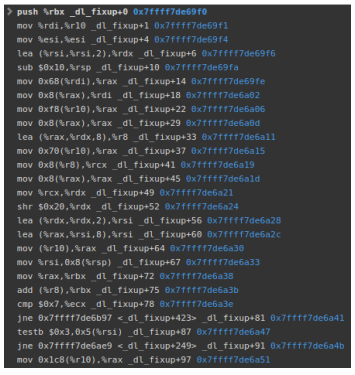


Рис. 7. Дизассемблированный код

Слева располагается выпадающий список всех исходных файлов программы, доступных для отладки (рис. 8).

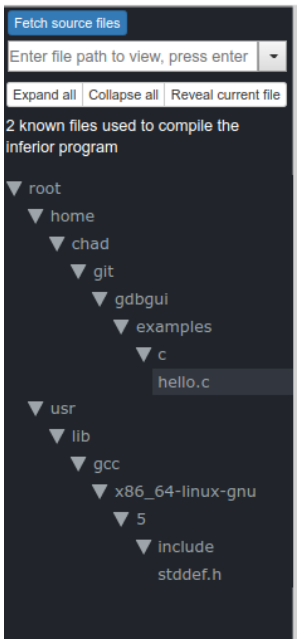


Рис. 8. Список исходных файлов

С правой стороны также находится несколько разделов.

Раздел «Threads» содержит информацию об отлаживаемых потоках и их стеке вызовов. Там же можно выбрать нужный поток для отладки.

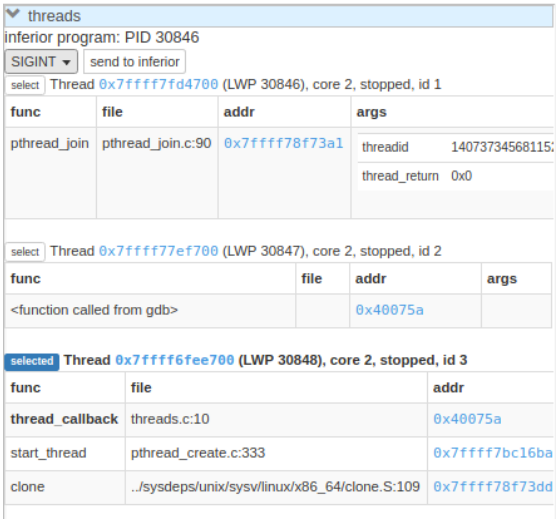


Рис. 9. Список потоков

Под ним располагается раздел с информацией о локальных переменных (рис. 10).

Рис. 10. Список локальных переменных

Также текущее значение переменной можно посмотреть, просто наведя на ее имя мышку (рис. 11).

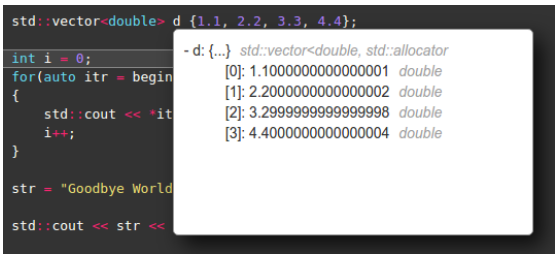


Рис. 11. Просмотр значения переменной

В разделе «Expressions», можно задать выражения, которые будут вычисляться при каждой остановке программы (рис. 12).

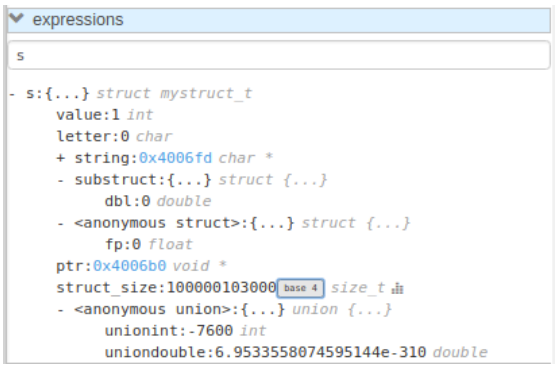


Рис. 12. Выражения

Вычисленные значения выражений могут отображаться на координатной сетке, позволяющей проследить историю изменений заданного выражения (рис. 13).

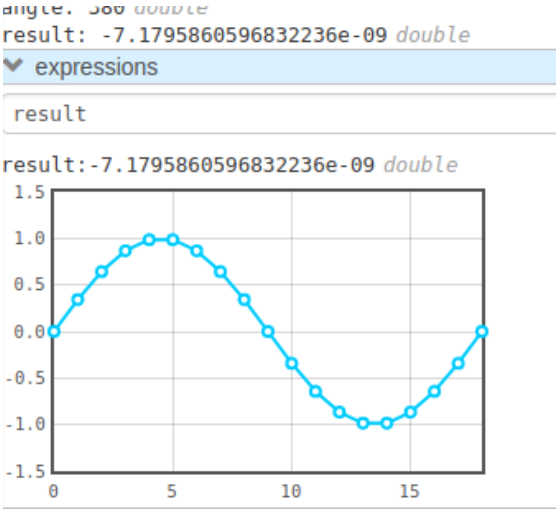


Рис. 13. Изменение значения выражения

Ниже располагается раздел, в котором можно посмотреть содержимое памяти отлаживаемой программы (рис. 14). Следует обратить внимание, что все шестнадцатеричные адреса gdbgui делает гиперссылками для удобства изучения памяти.

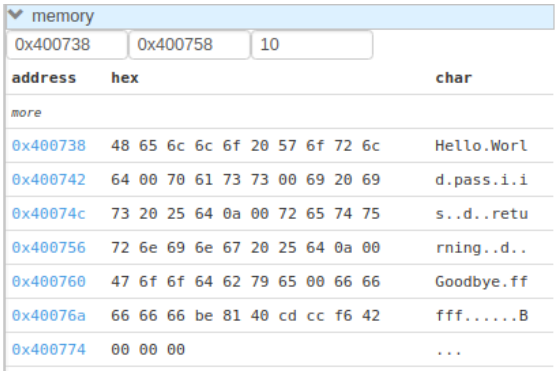


Рис. 14. Содержимое памяти

Далее находится раздел с регистрами текущего отлаживаемого потока (рис. 15). Все измененные с момента предыдущей остановки программы регистры отображаются желтым цветом.

Рис. 15. Регистры программы

На отдельной вкладке располагается панель управления сеансами gdbgui (так называемая dashboard, рис. 16).

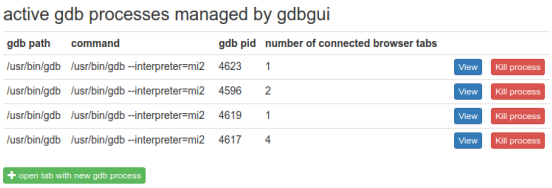


Рис. 16. Панель управления сеансами отладки

Разработчик имеет возможность переключаться между разными сеансами отладки, участвуя в них или активно (только один пользователь может вести сеанс отладки), или пассивно в качестве наблюдателя.

5. Заключение

Авторами была произведена доработка текущей версии gdbgui, исправлен ряд ошибок, касающийся, в частности, отображения регистров отлаживаемой программы. По результатам тестирования gdbgui его можно рекомендовать к использованию вместе с любым отладчиком, поддерживающим gdb/MI интерфейс.

Одним из главных плюсов можно выделить его независимость от инструментальной системы, на которой выполняется отладчик gdb, что несомненно облегчает удаленную отладку. Клиентом отладки может выступить любое устройство, имеющее веб-браузер с поддержкой typescript. Также следует упомянуть простоту кастомизации gdbgui, что позволяет его адаптировать под конкретные задачи разработки и отладки сложных систем. Кроме того, gdbgui дает разработчику возможность устроить совместный сеанс отладки с другими даже удаленными разработчиками, что увеличивает вероятность оперативного обнаружения и устранения ошибки.

«Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Исследование и реализация программной платформы для перспективных многоядерных процессоров» (FNEF-2022-002).»

Visual Debugging with gdbgui

Vladimir Galatenko, Konstantin Kostiukhin, Anastasiia Frolova

Abstract. The gdb interactive debugger is the main debugging tool for programs developed on the Linux platform. Providing many opportunities for debugging, gdb has one (according to a large number of developers) significant drawback – the command-line interface. Many developers still prefer more convenient graphical interfaces. In this article, we will consider a promising graphical frontend for gdb, which is a browser client, due to which users have the opportunity for platform-independent debugging.

Keywords: debugging, gdb, frontend, GUI

Литература

1. GDB: The GNU Project Debugger, https://www.sourceware.org/gdb
2. Галатенко В.А., Костюхин К.А. Машинно-ориентированный текстовый интерфейс отладчика GDB // Труды научно-исследовательского института системных исследований Российской академии наук, 2022, Т. 12, № 4, стр. 38–42.
3. GDBGUI: A browser-based frontend to gdb (gnu debugger), https://www.gdbgui.com

Простые генераторы отчетов для учетно-управленческих информационных систем

А.Б. Бетелин[1], И.Б. Егорычев[2], А.А. Прилипко[3], Г.А. Прилипко[4], С.Г. Романюк[5], Д.В. Самборский[6]

[1]ab@niisi.msk.ru, [2]egorychev@gmail.com, [3]aaprilipko@niisi.msk.ru, [4]prilipko@niisi.msk.ru, [5]sgrom@niisi.ras.ru, [6]samborsky_d@fastmail.com

[1,2,3,4,5,6]ФГУ ФНЦ НИИСИ РАН, Москва, Россия

Аннотация. В статье приводится описание функциональных возможностей и особенностей работы средств генерации отчетов, применяемых в учетно-управленческих информационных системах НИИСИ РАН.

Ключевые слова: генератор отчетов, информационная система, учет, управление, программное обеспечение, база данных

1. Введение

В современном мире, пожалуй, ни одна программная информационная система не может обойтись без генератора отчетов.

В контексте информационных систем отчетом принято считать представление информации, извлеченной из хранилища системы, в сформатированном виде, удобном для визуального восприятия человеком. Отчет может быть отображен на экране монитора, напечатан на принтере, выведен в файл и т.п.

Создание отчетов является одной из важных задач, возникающих при разработке различных информационных систем. Отчет включает, помимо содержательной информации, еще и некоторую оформительскую составляющую, формирование которой при помощи стандартных средств зачастую представляет собой весьма трудоемкую задачу. Поэтому в программных информационных системах решение этой задачи обычно возлагается на генератор отчетов — специальное программное обеспечение, позволяющее представить информацию в удобочитаемом структурированном виде [6]. На основе входных данных генератор формирует документ, который затем можно напечатать на бумаге или же сохранить в файл в каком-либо из стандартных форматов данных, применяемых для электронных документов.

В общем случае источником исходного материала для генератора отчетов могут служить файлы с данными, а также другие программы, программные системы и комплексы, осуществляющие ввод, обработку и хранение информации. В информационных системах данные извлекаются, как правило, из хранилища системы, представляющего собой набор файлов или же базу данных.

В настоящее время существует великое множество разнообразных генераторов отчетов, реализованных как в виде составной части программной системы (встроенные генераторы), так и в виде самостоятельной программы. В последнем случае генератор, как правило, снабжается программным интерфейсом, позволяющим воспользоваться его функциональностью из другой программы или программной системы.

Встроенные генераторы отчетов обычно используются в различных СУБД (например, Access [8], Informix [9]), а также в учетных системах семейства «1С:Предприятие» [7]. Среди самостоятельных программ формирования отчетов широко известны Crystal Reports [10], FastReport [11], Stimulsoft Reports [12]. Кроме упомянутых выше и многих других коммерческих продуктов в последние годы также получили распространение несколько программ генерации отчетов с открытым исходным кодом, например: YARG [14], Carbone [15], appy.pod [16]. Эти программы объединяет то, что они самостоятельно редактируют содержимое XML-данных, содержащихся в документах стандартов OpenDocument (ODF) [17] и Office Open XML (OOXML) [18], и избегают использования сред OpenOffice [19] и Microsoft Office [20].

В НИИСИ РАН в разное время был разработан ряд программных систем для автоматизации управленческого и бухгалтерского учета [1]. Эти системы снабжены средствами генерации отчетов, которые создавались с учетом следующих потребностей и условий:

— пользователем учетных систем является административно-технический персонал организации, не имеющий навыков программирования;

— все отчеты являются типовыми, каждый из них имеет заранее определенный вид (заголовок, наполнение, размещение информации на

печатной странице);

— не требуется создание отчетов в произвольной форме с возможностью индивидуального программирования;

— в отчетах не требуется наличие графиков, диаграмм, иллюстраций и т.п.; информация представляется исключительно в виде текста.

Общая схема функционирования разработанных инструментов создания отчетов состоит в следующем. Система с помощью встроенных средств для взаимодействия с базой данных извлекает из нее нужную для отчета информацию, обрабатывает ее и на этой основе формирует входной поток информации для генератора отчетов. Далее генератор отчетов принимает входной поток и, используя содержащуюся там информацию, преобразует шаблон отчета в итоговый документ, который выводит в файл заданного формата. В зависимости от применяемого генератора это может быть текстовый файл или же документ в формате электронной таблицы или текстового процессора сред OpenOffice и MS Office.

Настоящая статья посвящена описанию функциональных возможностей и особенностей работы рассматриваемых средств генерации отчетов.

2. Генератор отчетов системы БРОД

Система автоматизации бухгалтерского и налогового учета БРОД [1] была создана в НИИСИ РАН более 25 лет назад и продолжает развиваться и эксплуатироваться в комплексе со стандартными бухгалтерскими системами. Пользовательский интерфейс системы ориентирован на взаимодействие в стиле алфавитно-цифрового (а/ц) терминала. Соответственно, в оформлении отчетов применяются символы из имеющихся на клавиатуре а/ц терминала, дополненные набором символов псевдографики.

Использование стандартных средств генерации отчетов, предоставляемых языками 4gl [13], оказалось достаточно трудоемким и даже при небольших изменениях в отчетах каждый раз требовало модификации и пересборки программ. Поэтому более технологичным решением явилось создание специализированного комплекса программ, совместно решающих задачи как формирования отчетов, так и их последующего форматирования и печати. Этот комплекс включает программу генерации отчетов **tgf**, обеспечивающую получение текстовых отчетов (краткий обзор tgf приведен в [3]) и набор вспомогательных программ для дополнительной обработки файлов с отчетами.

2.1. Схема работы генератора и команды преобразования шаблона

Шаблон отчета находится в так называемом текстовом файле-форме, обычно с расширением .tgf. Он содержит образец внешнего вида отчета со специальными дополнительными полями, заключенными в квадратные скобки. В них находятся управляющие команды для генератора отчетов, которые могут задавать такие действия как перемещение по файлу-форме, удаление строк из входного потока, позиционирование их в шаблоне и т.п. Генератор считывает содержимое файла-формы и данные из входного потока, а затем, руководствуясь командами в управляющих полях, размещает считанные данные в шаблоне.

Запустить генератор из командной строки ОС Linux [22] можно, например, так:

```
tgf -f=template.tgf - -
```

Здесь template.tgf — имя файла-формы, входной поток поступает со стандартного ввода, а готовый отчет направляется на стандартный вывод.

Управляющие поля в шаблоне могут относиться к одному из двух типов.

Поле первого типа начинается с некоторого числа, за которым следуют одна или несколько однобуквенных команд:

[*число команда...команда*]

Число интерпретируется в зависимости от указанных за ним команд. Оно может обозначать либо номер строки из входного потока (строки нумеруются с 1), либо количество строк, которые надо удалить из входного потока, либо указание, на сколько строк надо переместиться по форме. Например:

l — удалить указанное число строк из начала входного потока;

f — переместиться вверх по файлу-форме на указанное число строк.

Это позволяет организовать циклы при выводе информации. Выход из цикла происходит либо при завершении поступления строк из входного потока, либо если первая строка при выполнении команды f начинается с указанного в последнем условном переходе проверочного слова-признака.

Помимо команд l и f генератор различает также следующие однобуквенные команды:

b — выводить строку с первого непустого символа;

e — вывести продолжение строки;

g — считать квадратную скобку входящей в поле для вывода информации;

n — не выводить данную строку в генерируемый отчет, если содержимое всех полей данной строки пусто;

c — разместить содержимое в центре поля (иначе содержимое прижимается к соответствующему краю поля).

Вот несколько примеров управляющих полей первого типа:

[2bg] — вставить 2-ю строку входного потока, предварительно удалив из нее передние пробелы, и выровнять ее по позиции левой квадратной скобки.

[2eng] — вставить продолжение 2-й строки, причем символ n внутри квадратных скобок указывает, что если продолжения нет, то эту строку нужно будет убрать из получаемого отчета. Таким образом можно создать отчет с числом строк, достаточным для того, чтобы целиком поместить в него строку из входного потока.

[5l] — удалить из входного потока первые пять строк.

[4f] — переместиться вверх по файлу-форме на четыре строки, если не встретилось слово-признак условного перехода.

Поля второго типа служат для реализации условных переходов внутри файла-формы. Такое поле содержит восклицательный знак, перед которым может быть указано число, и некоторое слово-признак, которое запоминается для дальнейшего использования командой f:

[*число*!*слово*]

Если первая строка входного потока начинается с указанного слова *«слово»*, то это вызовет перемещение вниз по файлу-форме на указанное *число* строк.

Рассмотрим следующий пример. Пусть в файле-форме последовательно встречаются управляющие поля [3!КонеЦ], [2l] и [1f] (см. рис. 1), и первая строка входного потока начинается со слова «КонеЦ».

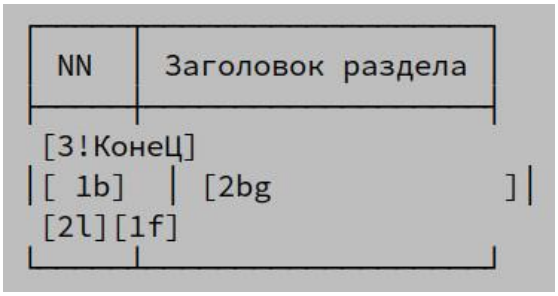


Рис. 1. Пример файла-формы

Тогда в процессе обработки поля [3!КонеЦ] выполняется перемещение вниз на три строки файла-формы с запоминанием слова «КонеЦ». Обработка поля [2l] приведет к удалению первых двух строк из входного потока. Если после этого первая строка входного потока будет начинаться со слова «КонеЦ», то при обработке поля [1f] произойдет смещение на следующую строку файла-формы, иначе будет выполнен переход вверх на одну строку.

Перечисленных механизмов оказалось вполне достаточно для изготовления текстовых отчетов любой сложности, необходимой при выпуске документов бухгалтерского и налогового учета.

Следует отметить, что с помощью генератора отчетов tgf можно решить и обратную задачу, т.е. извлечь информацию из файла с отчетом по некоторой форме, например, для печати содержимого его полей на специальных бланках. Здесь дополнительно используются возможности, предоставляемые программой pcp (см. подраздел 2.2.) по установке позиции на листе бумаге перед печатью необходимой информации.

2.1. Вспомогательные программы обработки текста

Довольно часто после генерации отчета возникает задача дополнительного форматирования находящихся внутри него таблиц. Таблица в данном случае представляется последовательностью текстовых строк, в которых содержимое одной ячейки таблицы отделяется от содержимого другой ячейки таблицы символом-разделителем (обычно для этого используется псевдографический символ вертикальной черты). Для дополнительной обработки таблиц в отчете был разработан ряд программ, функциональные возможности некоторых из них приведены ниже.

pcp — выполняет перекодировку файлов и/или подготовку их для печати на принтерах EPSON, HP и совместимых с ними.

tmk — выравнивает ширину столбцов таблицы, позволяя уменьшить их ширину до минимально необходимой, а также удалить пустые столбцы.

pff — разбивает отчет на страницы с учетом наличия в нем таблиц, вставляет дополнительные заголовки, осуществляет нумерацию страниц.

tpp — позволяет провести разбиение таблиц на страницы по ширине с возможностью учета границ столбцов и переноса нескольких первых столбцов таблицы на новые страницы.

Для того чтобы автоматически выполнять предварительное разбиение на страницы и дополнительное форматирование перед печатью, используются специальные файлы с расширением .ph, в которых указываются ключи, позволяющие перед печатью отчета вызвать необходимые программы с указанными ключами. Имя такого файла должно совпадать с расширением

файла с отчетом. Например, если файл с отчетом называется «report.txt», то специальный файл для него должен называться «.txt.ph».

Довольно часто при работе с информацией возникает задача обработки таблиц или CSV-файлов, полученных из файлов электронной таблицы MS Office, и переноса необходимой информации в систему или базу данных. Для этих целей была разработана программа tes, которая может выделить информацию из нужных столбцов таблицы, переставить их при необходимости, а также добавить пустые столбцы или столбец, содержащий номера строк исходного файла.

Все вышеназванные программы применяются в НИИСИ РАН с 1997 года. Они существенно облегчили решение целого ряда задач, связанных с разработкой и эксплуатацией системы расчета зарплаты, а также ряда других бухгалтерских и информационных систем. Эти программы входят в состав комплекса на основе текстового редактора **rk** [4,5] и поставляются вместе с ним в виде исходного кода на языке Си.

3. Генератор отчетов АСУ НИИСИ

Информационная система АСУ НИИСИ [1] была разработана в 2007 году на базе инструментальной среды с веб-интерфейсом [2] и в настоящее время продолжает развиваться. На текущий момент в АСУ НИИСИ насчитывается около 30 информационных подсистем для решения разного рода учетно-управленческих задач.

Взаимодействие пользователя с подсистемами АСУ НИИСИ осуществляется посредством графического веб-интерфейса, для которого устройством отображения информации служит графический монитор. Создаваемые в АСУ НИИСИ отчеты имеют внешнее оформление документов полиграфического качества, где возможно применение шрифтов различных гарнитур, кеглей, начертаний и т.п., а также таблиц с ячейками, обрамленными графическими линиями. Применяемый в АСУ НИИСИ генератор отчетов входит в состав ее базового инструментария.

3.1. Схема генерации отчета

Каждому отчету в АСУ НИИСИ соответствует отдельный командный SQL-файл, который содержит программу подготовки входного потока информации для генератора отчетов. Программа извлекает данные из базы данных АСУ и формирует строки входного потока. Система запускает этот SQL-файл на выполнение в начале процесса изготовления отчета.

Входной поток представляет собой последовательность текстовых строк, каждая из которых содержит либо команду, либо данные. Строки с командами начинаются с символа «#», а строки с данными такого префикса не имеют. В строках с командами в случае необходимости после символа «%» может присутствовать комментарий. Входной поток располагается в файле формата CSV.

Шаблоном отчета служит документ в формате ODF (ODS в случае электронной таблицы, или же ODT в случае текстового документа). В шаблоне с помощью переменных вида «#varname» обозначены позиции для вставки данных. Генератор считывает команды и данные из входного файла, в процессе выполнения этих команд преобразует шаблон и размещает данные на позициях, обозначенных переменными.

Отметим следующее различие между программами генерации отчетов системы БРОД и АСУ НИИСИ. В генераторе системы БРОД команды, управляющие процессом формирования отчета, размещаются в шаблоне, а входной поток содержит только данные для отчета. В генераторе АСУ НИИСИ, напротив, входной поток содержит и данные, и команды, шаблон же лишь описывает внешний вид отчета.

Кроме того, в отличие от системы БРОД схема работы генератора отчетов АСУ НИИСИ не предполагает перемещение по шаблону. Формирование отчета и подстановка значений переменных производится при помощи механизма работы с буферами. Буфер — это динамически создаваемое хранилище, в которое можно поместить одну или несколько строк из шаблона. Буфер создается автоматически, когда генератор встречает его имя во входном файле. Имя буфера может быть любым словом, содержащим буквы, цифры и символы подчеркивания.

Обычно процесс преобразования шаблона происходит следующим образом. Строки шаблона копируются в один или несколько буферов. Каждая строка, содержащая переменные, обязательно должна быть скопирована в буфер, чтобы присвоить им значения с помощью специальной операции заполнения буфера (см. подраздел 3.3). Из шаблона удаляются строки, не требующиеся в итоговом документе либо находящиеся не там, где необходимо. После этого содержимое буферов, в т.ч. строки с получившими значение переменными, копируется в шаблон в нужные позиции. Описание команд для выполнения указанных операций представлено в подразделе 3.3.

Готовый отчет выводится в файл в виде ODF-документа. При необходимости он может быть сконвертирован в формат электронной таблицы или текстового процессора MS Office с помощью программных средств OpenOffice.

3.2. Программа генерации отчета

Программа генерации отчета, обрабатываю-

щая шаблон и заполняющая его данными, написана на языке Perl и называется **xmlfilter.pl**. Она получает на входе указанный выше CSV-файл с командами и данными и перерабатывает файл content.xml из ODF-шаблона в аналогичный файл для готового отчета. Согласно спецификации формата OpenDocument [17] в файле content.xml находится содержательная часть ODF-документа. Отчет формируется при помощи специального скрипта, который сначала извлекает файл content.xml из ZIP-архива шаблона, затем запускает программу xmlfilter.pl, а в конце помещает переработанный файл content.xml в итоговый ZIP-архив отчета. Таким образом, в процессе создания ODF-документа отчета не используется ни программа текстового процессора OpenOffice, ни прикладные библиотеки этой программы. Это повышает надежность генератора отчета, т.к. опыт применения штатного способа формирования ODF-документов с помощью пакета SDK OpenOffice показал ненадежность работы этого ПО. Позже эти наблюдения подтвердились свидетельствами независимых разработчиков (например, см. описание истории создания генератора отчетов YARG [14]).

В командной строке ОС Linux программа генерации отчета может быть запущена так:

```
cat example.csv | ./bin/xmlfilter.pl \
--in src/content.xml \
--out res/content.xml
```

Здесь:

example.csv — входной файл в формате CSV;

src/content.xml — файл content.xml из шаблона отчета;

res/content.xml — переработанный файл content.xml для готового отчета.

3.3. Команды преобразования шаблона

Трансформация шаблона отчета в итоговый документ происходит в результате выполнения команд, наиболее важные из которых кратко описаны здесь с указанием формата вызова и примерами использования.

Аргументами команд могут быть номера строк шаблона либо имена буферов. В командах, оперирующих со строками, требуется указывать номер листа электронной таблицы и номер обрабатываемой строки (или же диапазон номеров строк). Эти параметры задаются следующим образом:

[*лист*:]*строка1*[*..строка2*][:o[ld]|n[ew]]

Листы электронной таблицы нумеруются натуральными числами, начиная с 0. Точно так же нумеруются строки шаблона. Если в шаблоне содержится ровно один лист, то его номер можно не указывать. Если шаблон является не электронной таблицей, а текстовым ODT-документом с таблицами, то вместо номера листа указывается номер таблицы. Вот несколько примеров указания номеров строк:

1 — строка с номером 1 (вторая строка шаблона);

2..4 — три строки с номерами 2, 3, 4;

1:0..9 — первые десять строк второго по порядку листа электронной таблицы (или же второй по порядку таблицы в ODT-документе).

Способ вычисления номера строки документа задается при помощи суффиксов «:old» (или «:o») и «:new» (или «:n»). Суффикс «:old» определяет позиции строк до выполнения команд вставки и удаления строк (см. ниже), а суффикс «:new» — после указанных операций. По умолчанию предполагается использование суффикса «:new». Например, выражение «5:old» указывает на позицию строки номер 5 до выполнения операций вставки и удаления.

Перейдем теперь к описанию команд.

«>» — **команда копирования**. Позволяет переписать одну или несколько строк из шаблона в буфер, содержимое буфера в итоговый документ или же содержимое одного буфера в другой. Формат вызова:

 # номера_строк | имя_буфера > номера_строк | имя_буфера

Примеры:

2 > BUF_1 — копируется строка номер 2 в буфер BUF_1;

3..4 > BUF_2 — копируются две строки с номерами 3 и 4 в буфер BUF_2;

BUF_2 > 7..8 — копируется содержимое буфера BUF_2 в две строки с номерами 7 и 8, при этом предыдущее содержимое строк 7 и 8 теряется;

BUF_2 > BUF_3 — копируется содержимое буфера BUF_2 в буфер BUF_3.

«>>» — **команда вставки**. Служит для размещения содержимого буфера в итоговом документе перед указанной строкой. Формат вызова:

 # имя_буфера >> номер_строки

Пример:

BUF_2 >> 2 — содержимое буфера BUF_2 помещается перед строкой номер 2.

«delete» — **команда удаления**. Дает возможность убрать из итогового документа одну или

несколько строк. Формат вызова:

delete *номера_строк*

Примеры:
delete 1 — удаляется строка номер 1;
delete 2..5 — удаляются строки с номерами от 2 до 5.

«fill» — команда заполнения буфера. Позволяет присвоить значение одной или нескольким переменным из шаблона. Каждой переменной может соответствовать одиночное значение (если нужно заполнить одну строку в итоговом документе) или же ряд значений (если нужно заполнить несколько строк). Формат вызова:

fill *имя_буфера*

Значения для присвоения переменным представляются в виде массива данных в CSV-формате, где в качестве разделителя значений разных колонок используется символ табуляции. В таком формате выдает данные клиент СУБД MySQL [21] в пакетном режиме. Если массив не пуст, то он должен состоять как минимум из двух строк. В первой строке помещается заголовок с именами колонок, которые должны совпадать с именами переменных в буфере. В последующих строках размещаются данные, соответствующие указанным в заголовке переменным. Порядок колонок может быть произвольным, допускается наличие лишних данных или же частичное их отсутствие. В простейшем случае в первой строке находится имя переменной, а во второй — ее значение. Для хранения этого массива предусмотрен специальный буфер, называемый *аккумулятором*.

Строки с переменными предварительно копируются из шаблона в буфер с именем *«имя_буфера»*. Команда fill выполнит подстановку значений переменных в буфере, используя данные из аккумулятора. Если аккумулятор содержит несколько строк данных, то для каждой такой строки будет создан и заполнен данными из этой строки отдельный экземпляр всех строк буфера. Например, если в буфере находятся две строки шаблона, а в аккумуляторе — три строки данных, то результатом выполнения команды fill будут шесть строк.

Если для некоторых переменных в строках, находящихся в буфере, в массиве данных не задано значений (нет соответствующих колонок), то значения этих переменных останутся неопределенными.

В массиве данных на любой позиции может использоваться специальное значение COUNT. При выполнении команды fill оно заменяется на номер текущей строки данных в массиве.

После заполнения данными содержимое буфера, как правило, копируется в итоговый документ с помощью команды «>» или «>>».

При работе с ODT-шаблоном имеется своя специфика, состоящая в том, что переменные, описанные вне таблиц, недоступны для обычных команд. Этим переменным можно присвоить значения только при помощи команды fill global, где global — имя особого буфера, обозначающего весь ODT-документ.

«reset» — команда сброса аккумулятора. Служит для очистки аккумулятора от содержащихся в нем данных. Формат вызова:

reset

«continue» — команда дополнения содержимого аккумулятора. Формат вызова:

continue

Иногда бывает удобно выводить массив данных не целиком, а отдельными порциями путем последовательного вызова нескольких операторов SELECT. Для этого предусмотрена команда continue, позволяющая продолжить заполнение аккумулятора очередной порцией данных.

Следующий пример показывает, как представляется во входном потоке массив данных, выведенный двумя порциями:

```
#% первая порция массива данных:
n1    n2    n3
1     1     1
2     4     8
# continue
#% вторая порция массива данных:
n1    n2    n3
3     9     27
```

«default» — команда определения подразумеваемого содержимого аккумулятора. Формат вызова:

default
массив_данных

В некоторых случаях оператор SELECT может выдать пустой массив данных. Тогда аккумулятор тоже будет пуст, последующая команда fill не изменит содержимого буфера, а в итоговом документе окажутся переменные, значение которых не определено. Чтобы этого не произошло, предусмотрена возможность задать содержимое аккумулятора по умолчанию, представляющее собой массив данных, который будет использован командой fill в случае, если аккумулятор

пуст.

В следующем примере приведен фрагмент входного потока, содержащий команду сброса аккумулятора, массив данных, команду определения пустого массива данных в качестве подразумеваемого и, наконец, команду заполнения буфера:

```
# reset
<массив данных, возможно пустой>
# default
n1    n2    n3
# fill BUF
```

Команду сброса аккумулятора (reset) следует выполнить до вывода данных, т.к. в противном случае будет использовано содержимое заполненного ранее аккумулятора, а подразумеваемое содержимое не будет воспринято.

Желательно определить подразумеваемое содержимое для каждого потенциально пустого набора данных, поскольку незаполненные переменные буфера не удаляются автоматически ввиду того, что они впоследствии могут быть заполнены на второй итерации или же командой fill global.

Конечно, проблему пустого аккумулятора можно решить на уровне SQL, заранее определив число строк в выводимом массиве данных, и если он пуст, то не выполнять вывод данных.

«drop» — команда удаления листа электронной таблицы или таблицы из текстового документа. Дает возможность удалить указанный лист электронной таблицы или же таблицу из текстового документа. Формат вызова:

drop *номер_листа_или_таблицы*

«columns» — команда копирования листа электронной таблицы. Служит для копирования атрибутов колонок одного листа электронной таблицы в другой лист. Если необходимо скопировать еще и содержимое строк таблицы, то это придется сделать вручную. Формат вызова:

columns *номер_листа* > *номер_листа*

3.4. Особенности, обусловленные характером работы процессора XML

Сегменты шаблона, содержащие более чем одну пустую строку, в ODF XML заменяются одной специальной строкой, что сбивает нумерацию строк. Поэтому в шаблонах следует использовать не более одной пустой строки для разделения данных или же учитывать это обстоятельство, считая сегменты пустых строк за одну строку.

Имена переменных, занимающих всю ячейку и имеющих тип decimal или date, должны содержать суффиксы _n или _d соответственно. Например: #summa_n, #mdate_d. Это связано с тем, что процессору XML необходимо принудительно выставить тип ячейки.

3.5. Пример работы генератора отчетов

Рассмотрим пример отчета, в котором первые три натуральных числа (1, 2 и 3) выводятся в первой степени, в квадрате и в кубе. ODS-шаблон для этого отчета изображен на рис. 2.

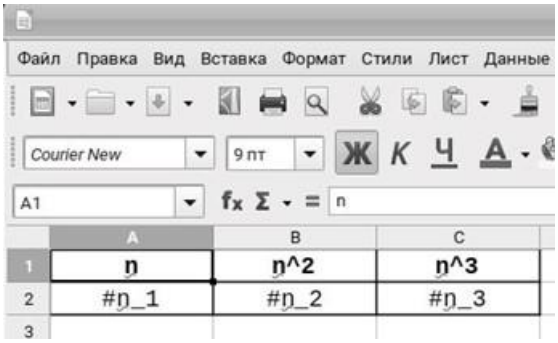


Рис. 2. Пример ODS-шаблона

Первая строка шаблона содержит заголовок отчета, а во второй находятся переменные #n_1, #n_2 и #n_3, которым будут присваиваться значения первой степени, квадрата и куба очередного числа.

Содержимое SQL-файла с программой подготовки входного потока для генератора отчетов может быть, например, таким:

```
SELECT '#% Генерация отчета' AS '#';
SELECT '# 1 > BUF' AS '#';
SELECT '# delete 1' AS '#';
SELECT '#% массив данных:' AS '#';
SELECT i AS 'n_1', i*i AS 'n_2', i*i*i
AS 'n_3' FROM t_num WHERE i <= 3;
SELECT '# fill BUF' AS '#';
SELECT '# BUF >> 1' AS '#';
```

Здесь путем вызова ряда операторов SELECT порождается последовательность команд и данных, образующих входной поток. После выполнения этой SQL-программы входной поток будет помещен в CSV-файл следующего вида:

```
#% Генерация отчета
# 1 > BUF
# delete 1
#% массив данных:
n_1   n_2   n_3
1     1     1
2     4     8
3     9     27
# fill BUF
# BUF >> 1
```

В первой строке файла находится комментарий. Затем идет команда копирования строки с номером 1 в буфер с именем BUF, а после нее — команда удаления строки номер 1 из шаблона. Далее после еще одной строки с комментарием определяется массив данных для присваивания переменным в буфере BUF. В следующей строке выполняется присваивание значений переменным в буфере BUF при помощи команды fill. Наконец, содержимое буфера BUF (переменные, получившие значения) вставляется в шаблон перед строкой номер 1, тем самым формируя итоговый документ.

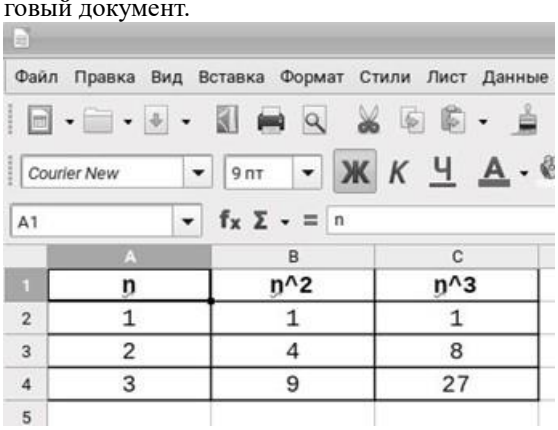


Рис. 3. Пример готового отчета

Далее этот файл отправляется на вход генератора, который в качестве результата своей работы создает файл content.xml для готового отчета в формате ODS, изображенного на рис. 3.

4. Заключение

Описанные в данной работе средства генерации отчетов и вспомогательные программы успешно применяются в НИИСИ РАН не только для создания всевозможных отчетов в учетно-управленческих системах. Они помогают в процессе подготовки отчетности в электронной форме в виде XML-файлов, а также при решении разнообразных задач обработки и преобразования текстовой информации и вопросов миграции данных между информационными системами.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН (Проведение фундаментальных научных исследований (47 ГП) по теме «1021060909180-7-1.2.1 Развитие методов математического моделирования распределенных систем и соответствующих методов вычисления. (FNEF-2022-0007)»).

Simple Report Generators for Accounting and Management Information Systems

A.B. Betelin, I.B. Egorychev, A.A. Prilipko, G.A. Prilipko, S.G. Romanyuk, D.V. Samborskiy

Abstract. The article describes the functionality and features of the report generation tools used in accounting and management information systems of SRISA RAS.

Keywords: report generator, information system, accounting, management, software, database

Литература

1. А.Б. Бетелин, И.Б. Егорычев, А.А. Прилипко, Г.А. Прилипко, С.Г. Романюк, Д.В. Самборский. Методы создания распределенного комплекса информационных систем для автоматизации управленческого и бухгалтерского учета. "Труды НИИСИ РАН", т.8 (2018), №4, 175-180.

2. И.Б. Егорычев. Инструментарий для построения автоматизированных учетных систем с WEB-интерфейсом. «Математическое и компьютерное моделирование систем: теоретические и прикладные аспекты», сб. науч. тр. НИИСИ РАН под ред. акад. В.Б.Бетелина. М., НИИСИ РАН, 2009, 81-90.

3. Г.А. Прилипко. Генератор отчетов и обработка текстовой информации. Достижения и приложения современной информатики, математики и физики. "Материалы Всероссийской научно-технической конференции", Уфа, РИЦ БашГу, 2012, 69-71.

4. Г.А. Прилипко. Исследование и разработка интерактивных языковоориентированных систем редактирования и визуализации программ. Автореферат кандидатской диссертации. МГУ им. М.В. Ломоносова, Москва, 1986, 23 с.

5. Г.А. Прилипко. Применение ЭВМ в учебном процессе. Опыт использования ЭВМ в обучении. "Межвузовский сборник научных трудов" под ред. О.М.Петрова. М., ВЗМИ, 1986. 26-30.

6. Генератор отчетов. https://ru.wikipedia.org/wiki/Генератор_отчетов (дата обращения 10.02.2023)

7. Отчеты в 1С. https://www.1s-up.ru/otchety-v-1s/ (дата обращения 10.02.2023)
8. Создание отчетов в Access. https://maxfad.ru/ofis/ms-access/462-sozdanie-otchetov-v-access.html (дата обращения 10.02.2023)
9. Язык программирования баз данных Informix-4GL. https://ami.nstu.ru/~vms/method5/posobie4_4gl.HTM (дата обращения 10.02.2023)
10. Сайт Crystal Reports. https://www.sap.com/products/technology-platform/crystal-reports.html (дата обращения 10.02.2023)
11. Сайт Fast Report. https://быстрыеотчеты.рф/ru/ (дата обращения 10.02.2023)
12. Сайт Stimulsoft Reports. https://www.stimulsoft.com/ru (дата обращения 10.02.2023)
13. Fourth-generation programming language. https://en.wikipedia.org/wiki/Fourth-generation_programming_language (дата обращения 10.02.2023)
14. YARG - open-source библиотека для генерации отчетов. https://www.jmix.ru/cuba-blog/report-generator/ (дата обращения 10.02.2023)
15. Сайт проекта Carbone. https://carbone.io/documentation (дата обращения 10.02.2023)
16. Сайт проекта appy.pod. https://appyframe.work/13 (дата обращения 10.02.2023)
17. OpenDocument technical specification. https://en.wikipedia.org/wiki/OpenDocument_technical_specification (дата обращения 10.02.2023)
18. Office Open XML. https://en.wikipedia.org/wiki/Office_Open_XML (дата обращения 10.02.2023)
19. OpenOffice.org. https://en.wikipedia.org/wiki/OpenOffice.org (дата обращения 10.02.2023)
20. Microsoft Office. https://en.wikipedia.org/wiki/Microsoft_Office (дата обращения 10.02.2023)
21. MySQL. https://en.wikipedia.org/wiki/MySQL (дата обращения 10.02.2023)
22. Linux. https://en.wikipedia.org/wiki/Linux (дата обращения 10.02.2023)

Применение технологии OpenCL для ускорения вычисления интегралов

А.А. Бурцев

ФГУ ФНЦ НИИСИ РАН, Москва, Россия, burtsev@niisi.msk.ru

Аннотация. Статья посвящена применению технологии OpenCL, позволяющей использовать мощные ресурсы графических процессоров для повышения быстродействия вычислительных программ. Рассматриваются варианты параллельных программ, разработанных для ускорения операции вычисления интегралов в среде OpenCL.

Ключевые слова: параллельное программирование, технология OpenCL, гетерогенные системы, численные методы вычисления интегралов

1. Введение

Современные высокопроизводительные системы для ускорения вычислительных программ, как правило, предлагают освоить те или иные особые технологии разработки параллельной программы, ориентированные на исполнение такой программы на многоядерных процессорах с общей памятью (OpenMP [1, п.5.1]) или на семействе компьютеров, связанных сетью (MPI [1, п.5.2]). Однако, сегодня можно ускорить выполнение своей вычислительной программы даже на одном компьютере, если в его составе имеется мощная видеокарта, которая поддерживает технологию OpenCL [2].

Технология OpenCL позволяет ускорить вычислительную программу за счёт её особого распараллеливания для последующего её исполнения на массиве однородных специализированных процессоров, функционирующих в составе современной графической видеокарты. Сначала аналогичную технологию (CUDA) предложила компания NVIDIA, потом − компания Apple, а Khronos Group приняла эту технологию за основу и довела до стандарта, назвав её **OpenCL** (**Open C**omputing **L**anguage).

Данная статья продолжает знакомство с технологией OpenCL, начатое автором в работах [3-5]. В них рассматривались приёмы разработки в среде OpenCL эффективных программ для решения отдельных задач линейной алгебры и цифровой обработки сигналов. В частности, рассматривались варианты построения по технологии OpenCL программ, ускоряющих операцию перемножения больших матриц, и программ, ускоряющих в среде OpenCL выполнение операции быстрого преобразования Фурье (БПФ для комплексных векторов большой длины вида $N=2^P$).

Полученный опыт разработки OpenCL-программ позволяет утверждать, что с помощью технологии OpenCL действительно можно существенно ускорить выполнение таких вычислительных программ, в которых требуется осуществлять однотипные операции над огромными массивами данных. Так, например, операцию перемножения больших матриц вещественных чисел (размером 2048×2048) удалось ускорить примерно в **15** раз (см. табл. 6 в [3]) на обычном настольном компьютере с универсальным процессором CPU Intel i59400 (с частотой 2.9 ГГц) и встроенным графическим процессором GPU UHD 630 (с частотой 350 МГц). А на компьютере с процессором CPU Intel i3-2100 (3.1 ГГц) и подключённой специализированной видеокартой NVidia GeForce 1050ti (1392 МГц) удалось добиться ускорения для той же операции в **95** раз (см. табл. 9 в [3]). На тех же компьютерах удалось значительно ускорить и операцию БПФ для комплексных векторов: например, для векторов длины $N=2^{24}$ получены показатели ускорения соответственно в **32** раза и в **190** раз (см. табл. 1 и табл. 2 в [4]).

Следует, однако, заметить, что для совершения в среде OpenCL операций над массивами приходится непроизводительно тратить время на передачу исходных данных из основной памяти компьютера в память OpenCL-устройства, а после – копировать обратно результаты совершённой операции. И поэтому можно надеяться, что для вычислительных задач, в которых такие передачи данных сведены к минимуму (или вообще отсутствуют), применение технологии OpenCL позволит обеспечить более значительные показатели ускорения. Приближённое вычисление интегралов как раз и относится к такой группе вычислительных задач, выполнение которых можно легко распараллелить, а передачи данных свести с минимуму.

2. Последовательная программа для вычисления интегралов

Необходимость приближённого вычисления интегралов возникает всякий раз, когда невозможно (или затруднительно) выразить требуемый интеграл аналитическим методом в виде законченной формулы с применением элементарных функций. Известен целый ряд подобных так называемых «неберущихся» интегралов (см. [6]). Среди них отметим интеграл Френеля:

$$\int sin(x^2)\,dx$$

а также интеграл Пуассона:

$$\int e^{-x^2}dx$$

В дальнейшем будем использовать формулы этих интегралов при демонстрации примеров применения программ, разработанных автором в среде OpenCL для приближённого вычисления интегралов.

Для вычисления интегралов применяются разнообразные численные методы. Самый простой из них – метод прямоугольников. Он позволяет для интеграла вида:

$$\int_A^B f(x)\ \ dx$$

получить приближённое значение по формуле:

$$S = h \times \sum_{k=0}^{N-1} f(x_k)\,,$$

где: $h = \frac{B-A}{N}$, $x_k = A + k \cdot h$,

вычислив значение интегрируемой функции лишь в нескольких точках заданного отрезка.

Точность вычисления по такой формуле определяется числом **N,** на которое разбивается отрезок интегрирования [A,B]. И значение интеграла вычисляется, как сумма площадей всех прямоугольников, образованных в результате такого разбиения. Предполагается, что для достижения лучшей точности следует задавать большее значение N.

В обычной (последовательной) программе алгоритм вычисления интеграла по формуле прямоугольников можно выразить такой Си-функцией:

```
real Intgrl(int N, real A, real B){
 int k;  real x, h, Sum;
  Sum=0.0; h= (B-A)/(real)N;
  for (k=0; k < N; k++ )
  { x = A + k*h; Sum= Sum + F(x); }
  return (Sum*h);
}//Intgrl
```

Предполагается, что в программе предварительно определяется тип **real** для представления вещественных чисел с одинарной:

```
#define real float
```

или двойной точностью:

```
#define real double
```

а также Си-функция **F** для вычисления интегрируемой функции. При этом формулу для вычисления такой функции удобно задавать как define-переменную:

```
#define FUNC sin(x*x) /* для Френеля */
//#define FUNC exp(-x*x) /* для Пуассона */
real F(real x) { return FUNC; }
```

Это позволяет быстро перенастроить всю программу на вычисление интеграла другой функции и/или для другого типа вещественных чисел.

3. Программа для вычисления интегралов в среде OpenCL

В среде OpenCL, где могут параллельно функционировать несколько вычислительных ядер в качестве исполнителей, можно обеспечить одновременное вычисление значений интегрируемой функции сразу в нескольких точках. Просуммировав затем эти значения, можно в итоге получить результат интегрирования значительно быстрее. Но для организации такого процесса параллельных вычислений должна быть построена соответствующая OpenCL-программа, т.е. программа, осуществляющая требуемое распараллеливание в среде OpenCL.

Всякая программа, разработанная в расчёте на её параллельное исполнение в среде OpenCL, состоит из основной программы (OpenCL-приложения на языке Си) и нескольких так называемых процедур ядра (на языке OpenCL). Основная программа запускается на основном процессоре (хосте), подготавливает OpenCL-среду и периодически запускает в ней приготовленные процедуры ядра параллельно сразу на всех её вычислительных узлах − так называемых обрабатывающих элементах (processing element).

Для подготовки OpenCL-среды необходимо проинициализировать OpenCL-платфому, дескрипторы OpenCL-устройств, приготовить OpenCL-контекст и создать в нём дескриптор очереди команд. Требуется также приготовить объекты для представления в памяти OpenCL-устройства данных, подлежащих обработке, дескриптор для каждой процедуры ядра, и особый объект, содержащий в откомпилированной форме код всех процедур ядра. Как обеспечить все эти подготовительные действия, подробно объяснялось в предыдущих статьях автора (см.

п. 2.3-2.5 в [3] и п.3.2 в [4]).

Здесь же сосредоточим внимание, главным образом, на процедурах ядра, отражающих специфику выполнения задачи интегрирования, и той части основной программы, которая обеспечивает их запуск и параллельное исполнение на массиве вычислительных ядер.

3.1. Процедуры OpenCL-ядра для вычисления интеграла

Вычисление значения интеграла в среде OpenCL выполним в два этапа. На первом этапе будем запускать на всех исполнителях процедуру ядра **_Intgrl** (см. далее), а на втором этапе запустим процедуру ядра **_Sum** лишь на одном исполнителе.

Процедуры ядра, предназначенные для исполнения в среде OpenCL на каждом обрабатывающем элементе (ОЭ), подготовим в предположении, что при их параллельном запуске становятся известны общее количество таких исполнителей **P**, а также количество **Q** и размер **WS** рабочих групп (Working Group), в которые их можно объединять для совместной работы (предполагается, что **P** делится нацело на **WS**).

И будем полагать, что в качестве параметров процедуре ядра можно передать границы отрезка интегрирования [**A**,**B**] и количество точек **N**, на которых следует вычислять значения интегрируемой функции. А формула вычисления предназначенной для интегрирования функции задаётся (как и в Си-программе) с помощью **define**-переменной **FUNC**:

```
#define FUNC sin(x*x) /* для Френеля */
//#define FUNC exp(-x*x) /* для Пуассона */
real F(real x) { return FUNC; }
```

На первом этапе каждый исполнитель, узнав свой глобальный номер **i** в массиве всех исполнителей, сам определяет количество и район расположения точек отрезка, назначенных ему для обработки. Для этого он определяет, сколько точек должен обрабатывать каждый исполнитель **n**=**N/P,** и с какой по порядку точки отрезка ему следует их отсчитывать **p**= **i**×**n**. (Заметим, что в случае, когда **N** не делится нацело на **P**, необходимо скорректировать вычисление **n**=**N/P+1** и количество обрабатываемых точек для последнего исполнителя **n**=**N**−**P**.)

Вычисленные во всех точках значения функции необходимо просуммировать. Пусть сначала каждый исполнитель просуммирует вычисленные им значения функции в назначенных ему точках. И полученную сумму **S** сразу умножит на величину шага **h=(B-A)/N**.

Эти итоговые результаты, полученные таким способом каждым исполнителем, снова необходимо просуммировать. И процесс такого суммирования тоже желательно распараллелить. Для этого объединим исполнителей (ОЭ) в рабочие группы. В локальной памяти каждой такой группы выделим место для массива (**L**), в который поручим каждому исполнителю этой группы записать полученный им итоговый результат на позицию, соответствующую его локальному номеру **j** (в этой группе). Дождёмся, когда все исполнители группы совершат такую запись. И после этого поручим одному из исполнителей группы (с номером **j**=0) просуммировать все значения массива **L** и записать итоговый результат всей группы в глобальный массив **Sum** на позицию, соответствующую номеру данной рабочей группы **g**.

Такой алгоритм действий оформим в виде процедуры ядра **_Intgrl**, которая будет запускаться на каждом исполнителе на 1-ом этапе:

```
__kernel void _Intgrl
(int N, real A, real B,
 __global real *Sum, __local real *L )
{ int i,k,n,p; real x,h,S,Ai ;
 P=get_global_size(0); // кол-во всех исполнителей
 i=get_global_id(0);// глоб.номер исполнителя
 j=get_local_id(0); //его локал.номер в Раб.Группе
 g=get_group_id(0); // номер Раб.Группы
 WS=get_local_size(0); // кол-во эл-тов в Раб.Группе
 n= N/P; if((N%P)>0) n=n+1;
// n= кол-во точек для каждого исполнителя
 p= i*n; // номер его стартовой точки
 if((n+p)>N) { n=N-p; if(n<0) n=0; }
//n=кол-во точек,обрабатываемых этим исполнителем
 h=(B-A)/N; S=0.0; Ai= A+p*h;
 for (k=0; k<n; k++)
 { x=Ai+k*h; S= S + F(x); }
 L[j]= S*h;
 barrier(CLK_LOCAL_MEM_FENCE);
 if (j==0) { S= L[0];
  for (k=1; k<WS; k++ ) S= S+L[k];
  Sum[g]= S;
 }//if j==0
}//_Intgrl
```

Для завершения вычисления значения интеграла в среде OpenCL осталось выполнить (на 2-ом этапе) процедуру ядра **_Sum**, чтобы просуммировать значения, которые были записаны в массив **Sum** на 1-ом этапе:

```
__kernel void _Sum (int K,
__global real *Sum,__global real *Res)
{ int i; real S; S= 0.0;
 for (i=0; i<K; i++) S= S+Sum[i];
 *Res= S;
}//_Sum
```

Эти процедуры ядра вместе с необходимыми макроопределениями сосредоточим в файле “Intgrl.cl”, строки которого будут анализироваться при компоновке программного кода ядра в основной программе.

3.2. Организация запуска процедур ядра в основной программе

Предположим, что в основной программе уже осуществлены все необходимые действия по подготовке OpenCL-среды и в глобальной памяти OpenCL-устройства приготовлены объекты для представления массива **Sum** и результата интегрирования **Res**:

```
cl_uint szM= sizeof(cl_mem);
cl_uint szR= sizeof(real);
cl_mem memSum=clCreateBuffer(context,
CL_MEM_READ_WRITE, Q*szR, NULL, NULL);
cl_mem memRes=clCreateBuffer(context,
CL_MEM_READ_WRITE, szR, NULL, NULL);
```

А также образован объект **prgrm**, содержащий в откомпилированной форме код всех процедур ядра, и для каждой процедуры ядра создан дескриптор:

```
cl_kernel knI, knS;
 knI=clCreateKernel (prgrm, "_Intgrl",NULL);
 knS=clCreateKernel (prgrm, "_Sum", NULL);
```

Теперь для выполнения в среде OpenCL операции вычисления интеграла оформим функцию **clIntgrl** (с такими же параметрами, как и у Си-функции **Intgrl**, описанной в п.2):

```
real clIntgrl(int N, real A, real B)
```

Для осуществления операции вычисления интеграла в среде OpenCL в теле этой функции предусмотрим последовательность действий из следующих 5-ти частей:

1. Назначение 5-ти фактических параметров для процедуры ядра **_Intgrl**:

```
cl_uint szI= sizeof(int);
clSetKernelArg(knI,0,szI,&N);
clSetKernelArg(knI,1,szR,&A);
clSetKernelArg(knI,2,szR,&B);
clSetKernelArg(knI,3,szM,&memSum);
clSetKernelArg(knI,4, szR*WS, NULL);
```

2. Запуск процедуры ядра **_Intgrl** в среде OpenCL на множестве из **P** исполнителей, сосредоточенных в рабочих группах по **WS** элементов в каждой, с ожиданием её завершения всеми исполнителями:

```
size_t gWS[1]={P}; size_t lWS[1]={WS};
cl_event evI;
clEnqueueNDRangeKernel(cmndQ,
 knI,1,NULL,gWS,lWS,0,NULL,&evI);
clFinish(cmndQ);
```

3. Назначение 3-х фактических параметров для процедуры ядра **_Sum**:

```
clSetKernelArg(knS,0,szI,&Q);
clSetKernelArg(knS,1,szM,&memSum);
clSetKernelArg(knS,2,szM,&memRes);
```

4. Запуск процедуры ядра **_Sum** на одном исполнителе с ожиданием её завершения:

```
size_t gWS[0]={1}; cl_event evS;
clEnqueueNDRangeKernel(cmndQ,
 knS,1,NULL,gWS,NULL,0,NULL,&evS);
clFinish(cmndQ);
```

5. Выгрузка полученного результата интегрирования из объекта **memRes**:

```
real IntRes;
clEnqueueReadBuffer(cmndQ,
 memRes,CL_TRUE,0,szR,&IntRes,0,0,0);
 return IntRes;
```

4. Результаты ускорения операции вычисления интегралов в среде OpenCL

Чтобы оценить, насколько удалось ускорить операцию вычисления интеграла с помощью применения технологии OpenCL, была составлена программа (на языке Си), которая выполняла операцию вычисления интеграла два раза. Сначала она вызывала функцию **Intgrl** для обычного (последовательного) вычисления интеграла основным процессором, а затем вызывала функцию **clIntgrl,** чтобы подготовить OpenCL-среду и выполнить в ней ту же операцию множеством параллельно функционирующих вычислительных ядер OpenCL-устройства.

Данная программа была разработана как консольное приложение в среде MS Visual Studio 2017. Она компоновалась с различными вариантами задания define-переменных **FUNC** и **real** с тем, чтобы настроить её на вычисление различных интегралов с одинарной и двойной точностью представления вещественных чисел.

Программа запускалась в различных вариантах компоновки на двух разных платформах. Под ОС Windows-10 на аппаратной конфигурации, содержащей основной процессор CPU Intel-i59400 с частотой 2.9 Ггц и встроенный графический процессор GPU UHD 630 с частотой 350 МГц («платформа **Intel**»). И под ОС Windows-7 на аппаратной конфигурации, содержащей процессор Intel i3-2100 с частотой 3.1 Ггц и видеокарту NVidia GeForce 1050ti с частотой 1392 Мгц («платформа **NVidia**»).

Варианты программы, скомпонованные для вещественных чисел одинарной точности, многократно прогонялись для вычисления представленных ранее интегралов Френеля и Пуассона на обозначенных отрезках с заданием различного количества точек **N**. При этом замерялись усреднённые показатели времени $\mathbf{T_{CPU}}$ и $\mathbf{T_{CL}}$, затраченные на выполнение функций **Intgrl** и **clIntgrl**, а также вычислялся коэффициент ускорения как отношение $\mathbf{T_{CPU}/T_{CL}}$.

Результаты ускорения, полученные OpenCL-программой вычисления интегралов для платформы Intel, представлены в таблицах 1-2.

Таблица 1. Ускорение операции вычисления интеграла Френеля в среде OpenCL на платформе Intel (для P=2048,WS=128 на [-5.0,+5.0] = 1.0558)

R	$N=2^R$	T_{CPU}	T_{CL}	$T_{Я}$	K1	K2
10	1024	0.047454	0.218957	0.023915	0.217	1.9843
11	2048	0.094304	0.229216	0.024582	0.411	3.8363
12	4096	0.187053	0.211167	0.025249	0.886	7.4083
13	8192	0.377172	0.212242	0.026499	1.777	14.233
14	16384	0.769306	0.235982	0.028582	3.260	26.916
15	32768	1.519438	0.240124	0.032749	6.328	46.397
16	65536	3.035782	0.248154	0.042666	12.23	71.152
17	131072	5.895864	0.275902	0.060915	21.37	96.788
18	262144	11.860596	0.309114	0.097832	38.37	121.23
19	524288	23.913322	0.387304	0.172248	61.74	138.83
20	1048576	47.874524	0.571722	0.321914	83.74	148.72
21	2097152	95.581944	0.959715	0.620829	99.59	153.96
22	4194304	188.21317	1.557385	1.218161	120.9	154.51
23	8388608	382.50166	2.754440	2.413906	138.9	158.46
24	16777216	774.74099	5.146215	4.833396	**150.5**	**160.29**

T_{CPU} – время исполнения на CPU функции **Intgrl**
T_{CL} – время исполнения функции **clIntgrl** в OpenCL
$T_{Я}$ – время исполнения OpenCL-ядер
(все времена даны в миллисекундах)
K1 – общий коэффициент ускорения = T_{CPU} / T_{CL}
K2 – «чистый» коэффициент ускорения = T_{CPU} / $T_{Я}$

Таблица 2. Ускорение операции вычисления интеграла Пуассона в среде OpenCL на платформе Intel (для P=2048,WS=128 на [-5.0,+5.0] = 1.7724)

R	$N=2^R$	T_{CPU}	T_{CL}	$T_{Я}$	K1	K2
10	1024	0.042415	0.221982	0.023332	0.191	1.8179
11	2048	0.083851	0.210888	0.023999	0.398	3.4939
12	4096	0.166911	0.211253	0.024332	0.790	6.8597
13	8192	0.334737	0.216543	0.025082	1.546	13.346
14	16384	0.677476	0.250684	0.026665	2.703	25.407
15	32768	1.380362	0.243700	0.029665	5.664	46.532
16	65536	2.653926	0.242956	0.035999	10.92	73.722
17	131072	5.475312	0.259368	0.055832	21.11	98.068
18	262144	10.887064	0.279966	0.073748	38.89	147.63
19	524288	21.416330	0.349006	0.123498	61.36	173.41
20	1048576	43.283448	0.432322	0.223498	100.1	193.66
21	2097152	87.152944	0.719005	0.427664	121.2	203.79
22	4194304	173.91973	1.167820	0.831829	148.9	209.08
23	8388608	344.49520	1.949570	1.642242	176.7	209.77
24	16777216	693.86598	3.610980	3.259735	**192.2**	**212.86**

T_{CPU} – время исполнения на CPU функции **Intgrl**
T_{CL} – время исполнения функции **clIntgrl** в OpenCL
$T_{Я}$ – время исполнения OpenCL-ядер
(все времена даны в миллисекундах)
K1 – общий коэффициент ускорения = T_{CPU} / T_{CL}
K2 – «чистый» коэффициент ускорения = T_{CPU} / $T_{Я}$

Из этих таблиц видно, что с помощью применения технологии OpenCL операцию вычисления интегралов удаётся ускорить ещё в большей степени, чем операции перемножения матриц и быстрого преобразования Фурье. Так, например, вычисление интеграла Френеля на отрезке [-5.0,+5.0] с разбиением его на $N=2^{24}$ частей в среде OpenCL ускоряется в **150** раз, а вычисление интеграла Пуассона – в **192** раза. Для сравнения напомним, что умножение матриц с таким же количеством элементов ($2048\times2048=2^{24}$) на платформе Intel удавалось ускорить лишь в 15 раз, а операцию быстрого преобразования Фурье (БПФ) для векторов длины $N=2^{24}$ – в 32 раза.

А с применением специализированной видеокарты операцию вычисления интегралов можно ускорить ещё в большей степени, что подтверждают результаты прогона той же OpenCL-программы для платформы NVidia, представленные в таблицах 3-4.

Таблица 3. Ускорение операции вычисления интеграла Френеля в среде OpenCL на платформе NVidia (для P=2048,WS=128 на [-5.0,+5.0] = 1.0558)

R	$N=2^R$	T_{CPU}	T_{CL}	$T_{Я}$	K1	K2
10	1024	0.060003	0.201311	0.009088	0.298	6.6024
11	2048	0.120007	0.216712	0.008384	0.554	14.314
12	4096	0.240013	0.217312	0.007744	1.104	30.993
13	8192	0.490028	0.217512	0.008608	2.253	56.927
14	16384	0.960056	0.221812	0.009856	4.328	97.408
15	32768	1.920110	0.210412	0.011200	9.125	171.44
16	65536	3.880222	0.233813	0.012384	16.59	313.32
17	131072	7.860450	0.227613	0.017728	34.53	443.39
18	262144	15.540888	0.240213	0.027136	64.69	572.70
19	524288	31.001774	0.244814	0.044224	126.6	701.02
20	1048576	61.963544	0.296616	0.080928	208.9	765.66
21	2097152	124.00709	0.368021	0.127744	336.9	970.75
22	4194304	248.56421	0.465526	0.228480	533.9	1087.9
23	8388608	496.12838	0.704540	0.449536	704.2	1103.6
24	16777216	995.85696	1.197568	0.891712	**831.6**	**1116.8**

Таблица 4. Ускорение операции вычисления интеграла Пуассона в среде OpenCL на платформе NVidia (для P=2048,WS=128 на [-5.0,+5.0] = 1.7724)

R	$N=2^R$	T_{CPU}	T_{CL}	$T_{Я}$	K1	K2
10	1024	0.050003	0.213812	0.008480	0.234	5.8966
11	2048	0.110006	0.211612	0.008928	0.520	12.322
12	4096	0.230013	0.213912	0.008480	1.075	27.124
13	8192	0.450025	0.208111	0.009184	2.162	49.001
14	16384	0.960054	0.216012	0.007904	4.444	121.46
15	32768	1.820104	0.226412	0.009344	8.039	194.79
16	65536	3.600206	0.230813	0.010208	15.59	352.68
17	131072	7.160410	0.220812	0.011488	32.43	623.29
18	262144	14.280818	0.210812	0.014400	67.74	991.72
19	524288	28.961656	0.249014	0.021504	116.3	1346.8
20	1048576	57.323280	0.264815	0.033792	216.5	1696.4
21	2097152	115.50661	0.277015	0.060416	416.9	1911.8
22	4194304	231.16323	0.280016	0.095520	825.5	2420.1
23	8388608	461.52640	0.399522	0.152544	1155	3025.5
24	16777216	925.10291	0.521029	0.295296	**1775**	**3132.8**

В этих таблицах отмечено, что вычисление интеграла Френеля на том же отрезке [-5.0,+5.0] с разбиением на $N=2^{24}$ частей в среде OpenCL ускоряется в **831** раз, а вычисление интеграла Пуассона – в **1775** раза. Эти результаты можно сравнить с показателями ускорения на той же платформе (NVidia) операции перемножения

матриц такого же размера (2^{24}) и операции быстрого преобразования Фурье для векторов той же длины, упомянутых ранее (95 и 190 раз).

5. Заключение

Полученные результаты практически подтверждают, что операция вычисления интегралов, не требующая перекачки большого объёма данных между основной памятью и памятью OpenCL-устройства, в лучшей степени пригодна для её ускорения с помощью применения технологии OpenCL.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН «Проведение фундаментальных научных исследований (47 ГП)» по теме № FNEF-2022-0004 «Разработка архитектуры, системных решений и методов для создания микропроцессорных ядер и коммуникационных средств семейства систем на кристалле двойного назначения», Рег. № 122041100063-0.

Applying OpenCL Technology to Accelerating the Calculation of Integrals

A.A. Burtsev

Abstract. The article focuses on the use of OpenCL technology, which allows you to use powerful GPU resources to improve the performance of computing programs. Variants of parallel programs designed to accelerate the operation of calculating integrals in the OpenCL environment are considered.

Keywords: parallel programming, OpenCL technology, heterogeneous systems, numerical methods for calculating integrals.

Литература

1. В.В. Воеводин, Вл.В. Воеводин. Параллельные вычисления. Спб., БХВ-Петербург, 2004.

2. Официальный OpenCL–сайт организации Khronos Group, http://www.khronos.org/opencl/

3. А.А. Бурцев. Оптимизация операции перемножения матриц на основе технологии OpenCL. «Труды НИИСИ РАН», Т. 10 (2020), № 5-6, 100–112.

4. А.А. Бурцев. Ускорение быстрого преобразования Фурье на основе технологии OpenCL. «Труды НИИСИ РАН», Т. 11 (2021), № 4, 27–37.

5. А.А. Бурцев. Оптимизация операции быстрого преобразования Фурье в среде OpenCL. // «Труды НИИСИ РАН», Т.12 (2022), №1-2, 11-27.

6. Неберущиеся интегралы, https://www.matematicus.ru/vysshaya-matematika/integralnoe-ischislenie/neberushhiesya-integraly

Вычисление углов Тейта-Брайана ориентации трекера HTC VIVE

И.П. Саблин[1], М.В. Михайлюк[2], Д.В. Омельченко, Д.А. Кононов[3], Д.М. Логинов

[1,2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, [3]ИПУ РАН, Москва, Россия,
[1]sablinivan97@gmail.com, [2]mix@niisi.ras.ru

Аннотация. В работе рассматривается подключение, настройка и технология работы с трекером HTC Vive Tracker 2.0. Также предложен метод перевода матрицы перехода, получаемой от трекера, в углы Тейта-Брайана, с учетом обхода проблемы «блокировки кардана».

Ключевые слова: система виртуального окружения, HTC Vive Tracker, SteamVR, OpenVR API, углы Тейта-Брайана

1. Введение

В настоящее время одним из важных направлений исследований является разработка технологий, методов и алгоритмов создания и использования виртуальной реальности (VR) при помощи компьютерных технологий.

VR технологии успешно применяются в различных сферах жизнедеятельности. Их используют для создания систем виртуального окружения, тренажеров различных видов (космических, авиационных, медицинских и др.), виртуальных лабораторий, игровой индустрии, виртуальных выставок и т.д. Системы виртуального окружения включают подсистему создания виртуальных трехмерных сцен, подсистему расчета динамики виртуальных объектов и процессов, подсистему визуализации и подсистему управления динамическим объектами и процессами. Эти системы можно расширить до комплексов виртуальной реальности, добавив специальные устройства: шлемы (для отображения стерео визуализации), контроллеры (для управления объектами), трекеры (для получения их координат и ориентаций и передачи их в виртуальную среду) и др. Если прикрепить трекер к реальному объекту, то позиция и ориентация этого объекта также может быть вычислена и передана в систему виртуального окружения. Одним из таких устройств является HTC VIVE Tracker.

Данный трекер используется для широкого круга задач. В работе [1] он рассматривается как бюджетная замена костюма отслеживания движений тела. В ходе исследований установлено, что задержка в передаче данных из трекера составляет около 7 миллисекунд, что удовлетворяет требованиям систем виртуального окружения. В статье [2] проверяется точность определения позиции данного трекера, установленного на мобильном роботе. В ходе испытаний установлено, что HTC VIVE Tracker соответствует необходимой точности определения траектории робота. Авторы работы [3] использовали HTC VIVE Tracker в автоматизированном тесте на способность человека держать статический и динамический балансы (Berg Balance Scale или BBS) и показали его эффективность как инструмента скрининга, который обладает преимуществами низкой стоимости, высокой мобильности и простой настройки. В исследовании [4] на основе данных от трекеров были предложены два метода описания кинематики суставов нижних конечностей человека. Недорогая система позволяла собирать полные кинематические данные нижней конечности, получая непрерывные траектории трекера без трудоемкой калибровки системы. Полученные данные могут служить справочными данными для нормальной походки. Примером использования трекера для создания тренажеров может служить статья [5], в которой исследователи прикрепили трекеры HTC VIVE на симулятор велосипеда, и на основе передаваемых данных разработали тренажер для обучения безопасному передвижению в трафике.

Программный интерфейс трекера (OpenVR API) позволяет в любой момент времени получить его положение и ориентацию в виде матрицы перехода размером 3 x 4. Однако для многих задач более удобным является представление ориентации в виде углов Эйлера или Тейта-Брайана. Это позволяет, например, сократить число переменных, задающих ориентацию, с девяти (в матрице), до трех (в углах). HTC VIVE Trackers можно использовать и без дорогостоящего VR оборудования (шлема и контролеров). Целью данной работы является настройка HTC VIVE Tracker без использования дополнительной VR гарнитуры и вычисление углов Тейта-

Брайана из матрицы перехода.

2. HTC VIVE Tracker

HTC VIVE Tracker – это трекер виртуальной реальности, разработанный компанией HTC (см. рис. 1.). В состав комплекта входит сам трекер и специальное передающее устройство Dongle. Время автономной работы трекера до 6 часов.

Рис. 1. HTC VIVE Tracker

Помимо этого, для использования трекера необходимы базовые станции HTC VIVE (см.

Рис. 2. HTC VIVE Base Station 1.0

рис. 2.).

Для отслеживания положения и ориентации в пространстве данный трекер использует метод Lighthouse tracking, в котором базовые станции являются так называемыми «маяками». Каждая станция состоит из двух инфракрасных лазеров и массива светодиодов. Линейный лазер в любой момент времени излучает направленный плоский световой треугольник. Первый лазер меняет направление этого треугольника вокруг горизонтальной оси базовой станции, а второй – вокруг вертикальной оси. На самом же трекере располагаются фотодиоды. Вспышка светодиодов на базовой станции дает трекеру сигнал о начале временного интервала сканирования горизонтального лазера. Следующая вспышка дает сигнал для вертикального лазера. Если лазер попадает на фотодиод трекера, то трекер по времени этого попадания и ранее зафиксированного начала интервала вычисляет угол поворота плоскости лазера. Пересечение плоскостей от двух лазеров определяет прямую, а пересечение прямых от двух базовых станций определяет точку расположения фотодиода. Эти вычисления, проведенные для всех фотодиодов на трекере, позволяют точно и однозначно вычислить его позицию и ориентацию [6, 7].

3. Подключение и настройка HTC VIVE Tracker

Подключение трекера к компьютеру производится через среду SteamVR, которая обеспечивает работу приложений виртуальной реальности. Для установки данной среды необходимо найти в Интернет и скачать приложение Steam, которое является онлайн-сервисом цифрового распространения компьютерных игр и программ, разработанным компанией Valve в 2003 году. После установки и запуска приложения Steam необходимо в нем зарегистрировать пользователя, перейти во вкладку «МАГАЗИН», найти в поиске приложение SteamVR и установить его на компьютер.

Несмотря на то, что в данной работе используется только трекер, функционал Steam VR требует указание наличия шлема виртуальной реальности (HMD). Для этого необходимо перейти в директорию: steam\steamapps\common\SteamVR\drivers\null\resources\settings, открыть при помощи блокнота файл default.vrsettings и установить для параметра «enable» значение true (см. рис. 3). Аналогично, в директории steam\steamapps\common\SteamVR\resources\settings надо изменить файл default.vrsettings, а именно (см. рис. 4):

- установить значение false для параметра "requireHmd";
- установить значение null для параметра “forcedDriver";
- установить значение true для параметра "activateMultipleDrivers".

Следующим шагом будет установка базовых станций. На официальном сайте компании VIVE есть ряд советов и рекомендаций для правильного расположения этих станций и подготовки помещения [8]. В частности, станции можно установить на треноги на некотором расстоянии

друг от друга, направить их на участок части пространства, отслеживаемого трекером, и включить в электрическую сеть. Тренога с базовой станцией показана на рис. 5. Синхронизация базовых станций может осуществляться с помощью кабеля или оптически. Так как мы рассматриваем оптическую синхронизацию, то необходимо для одной базовой станции выбрать

```
"steamvr": {
   "requireHmd": false,
   "forcedDriver": "null",
   "forcedHmd": "",
   "displayDebug": false,
   "debugProcessPipe": "",
   "enableDistortion": true,
   "displayDebugX": 0,
   "displayDebugY": 0,
   "allowDisplayLockedMode": false,
   "sendSystemButtonToAllApps": false,
   "loglevel": 3,
   "ipd": 0.063,
   "ipdOffset": 0.0,
   "background": "",
   "backgroundUseDomeProjection": false,
   "backgroundCameraHeight": 1.6,
   "backgroundDomeRadius": 0.0,
   "environment": "",
   "hdcp14legacyCompatibility": false,
   "gridColor": "",
   "playAreaColor": "",
   "showStage": false,
   "drawTrackingReferences": true,
   "showGridCircles": true,
   "activateMultipleDrivers": true,
   "usingSpeakers": false,
```

Рис. 3. Фрагмент файла default.vrsettings.

```
"driver_null": {
    "enable": true,
    "loadPriority": -999,
    "serialNumber": "Null Serial Number",
    "modelNumber": "Null Model Number",
    "windowX": 0,
    "windowY": 0,
    "windowWidth": 2160,
    "windowHeight": 1200,
    "renderWidth": 1512,
    "renderHeight": 1680,
    "secondsFromVsyncToPhotons": 0.01111111,
    "displayFrequency": 90.0
}
```

Рис. 4. Фрагмент файла default.vrsettings.

режим работы «B», а для другой - «C» с помощью кнопки, расположенной на задней стороне.

Сам трекер автоматически подключается к адаптеру с электронным ключом (Dongle, см. рис. 6), который, в свою очередь, присоединяется к компьютеру с помощью USB кабеля. Для каждого трекера необходим свой dongle, поэтому использование нескольких трекеров может быть ограничено количеством USB портов. Для того, чтобы убедиться, что компьютер видит все подключенные устройства (включая искусственно воссозданный шлем), у SteamVR есть индикатор состояния. Если все подключено верно, то на индикаторе будут гореть четыре иконки: для трекера, шлема и двух базовых станций.

Рис. 5. Базовая станция на треноге

Рис. 6. Адаптер

Для настройки трекера нужно переопределить его роль, так как автоматически он определяется системой как контролер правой руки. Для этого необходимо перейти в настройки SteamVR, далее на вкладку «контролеры» - «управление трекерами VIVE» (см. рис. 7) и указать, что трекер не имеет определенной роли (Роль трекера = ОТКЛЮЧЕНО).

Также для более удобной работы в будущем, приложение Steam можно поставить в автозагрузку и при дальнейшем использовании трекера, достаточно будет просто включить его, а приложение SteamVR запуститься автоматически.

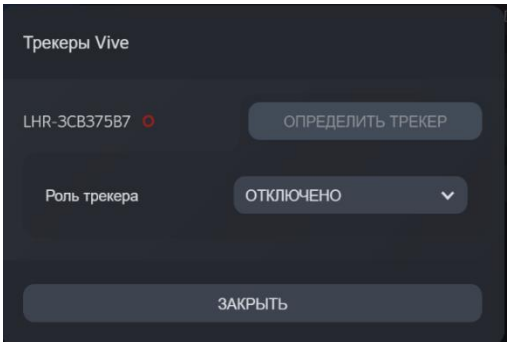


Рис. 7. Настройка роли трекера.

4. OpenVR

OpenVR – это открытый набор средств (SDK - software development kit и API - application programming interface) для последующей разработки приложений виртуальной реальности, предоставленный компанией Valve. При помощи OpenVR можно взаимодействовать с устройствами VR различных поставщиков [9]. Предоставленная библиотека является набором классов и виртуальных функций, написанных на языке C++. Вся дополнительная информация, а также сама библиотека OpenVR, находятся на официальном аккаунте корпорации Valve в GitHub [10], откуда ее можно скачать и разархивировать.

Для подключения данной библиотеки к проекту в среде Microsoft Visual Studio 2019 необходимо прописать путь к файлу openvr.lib в настройках Additional Include Directories, Additional Library Directories и Additional Dependencies. Кроме того, в одном из файлов проекта с помощью #include указывается директория, в которой расположен openvr.lib.

Для работы с системой виртуального окружения сначала необходимо инициализировать сам API с помощью функции VR_Init(). В OpenVR можно работать одновременно с k_unMaxTrackedDeviceCount = 64 устройствами.

Для определения номера слота, в который система подключила наш трекер, необходимо в цикле от 0 до k_unMaxTrackedDeviceCount-1 с помощью функции IsTrackedDeviceConnected проверить, что к i-му слоту подключено какое-то устройство, а с помощью функции GetTrackedDeviceClass – что это устройство является трекером и запомнить номер этого слота. По умолчанию в OpenVR первый слот занимает шлем виртуальной реальности, поэтому для

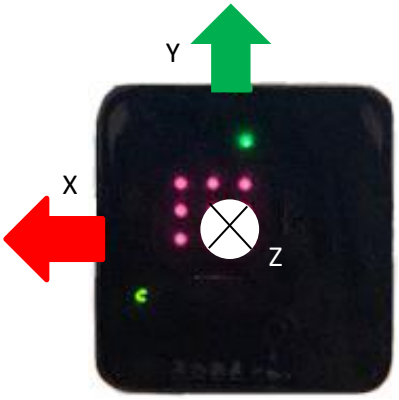


Рис. 8. Мировая система координат

определения слота трекера необходимо перебрать все остальные слоты. необходимо установить значение второго подключенного слота за трекером.

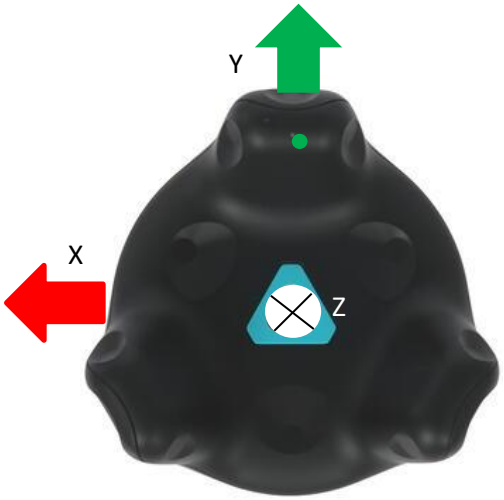


Рис. 9. Система координат трекера

В качестве мировой системы координат выбирается правосторонняя система координат базовой станции, работающей в режиме «С». Начало этой системы координат расположено в центре базовой станции, а оси направлены как показано на рис. 8. Начало правосторонней системы координат трекера расположено в его центре, а оси направлены как показано на рис. 9. Здесь важно, что ось Y направлена на индикатор зарядного устройства трекера. Для получения

матрицы преобразования из системы координат трекера в мировую систему координат можно использовать функцию GetControllerStateWithPose, первый параметр которой указывает тип выбранной системы координат, а четвертый параметр возвращает структуру типа TrackedDevicePose_t, содержащую матрицу перехода из системы координат трекера в эту систему координат.

5. Вычисление позиции и углов Тейта-Брайана трекера

Матрица перехода размерности 3 x 4, имеет в OpenVR тип HmdMatrix34_t. Ее четвертый столбец задает координаты позиции трекера в метрах, а первые три столбца – матрицу поворота, из которой нам необходимо вычислить углы Тейта-Брайана.

Рассмотрим в трехмерном пространстве две правосторонние (левосторонние) ортонормированные системы координат K_1 и K_2, центры которых совпадают, а оси направлены произвольно. Известно, что систему K_1 можно перевести с помощью трех поворотов вокруг ее осей в систему K_2. Если повороты осуществляются вокруг произвольных двух осей (например, XYX), то углы поворотов называются углами Эйлера. В зависимости от выбора пар осей можно получить шесть различных наборов углов Эйлера. Если же повороты выполняются вокруг трех различных осей (например, ZXY), то говорят об углах Тейта-Брайана. Их также существует шесть видов, в зависимости от выбора порядка осей поворота [11]. Кроме того, в силу двойственности, поворот вокруг некоторой оси координат на произвольный угол φ эквивалентен повороту всего пространства вокруг этой оси на тот же угол, только в противоположную сторону, т.е. на угол (-φ). При этом система координат остается неподвижной. Здесь мы как раз и будем рассматривать поворот пространства вокруг осей ZXY именно в этом порядке, т.е. сначала поворот вокруг оси Z, затем вокруг оси X и, наконец, вокруг оси Y. Порядок поворота можно указать с помощью индекса, записав это в виде $Z_\alpha X_\beta Y_\gamma$

Выведем матрицу поворота пространства вокруг оси Z. Для этого возьмем произвольный вектор V в плоскости XY. После поворота на угол α он перейдет в вектор V' (см. рис. 10).

Рассчитаем координаты вектора V':

$$x' = r\,\mathrm{Cos}(\theta-\alpha) = r\,\mathrm{Cos}\,\theta\,\mathrm{Cos}\,\alpha + rSin\theta\,\mathrm{Sin}\,\alpha =$$
$$= x\,\mathrm{Cos}\,\alpha + y\,\mathrm{Sin}\,\alpha$$
$$y' = r\,\mathrm{Sin}(\theta-\alpha) = r\,\mathrm{Sin}\,\theta\,\mathrm{Cos}\,\alpha - r\,\mathrm{Cos}\,\theta\,\mathrm{Sin}\,\alpha =$$
$$= y\,\mathrm{Cos}\,\alpha - x\,\mathrm{Sin}\,\alpha$$
$$z' = z$$

или

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \mathrm{Cos}\,\alpha & \mathrm{Sin}\,\alpha & 0 \\ -\mathrm{Sin}\,\alpha & \mathrm{Cos}\,\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

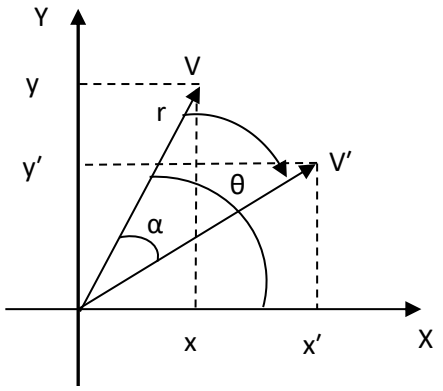


Рис. 10. Поворот вектора V в плоскости XY

Таким образом, матрица поворота пространства вокруг оси Z имеет вид

$$R_Z(\alpha) = \begin{pmatrix} \mathrm{Cos}\,\alpha & \mathrm{Sin}\,\alpha & 0 \\ -\mathrm{Sin}\,\alpha & \mathrm{Cos}\,\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Аналогично для поворотов вокруг осей X и Y матрицы поворота будут следующими:

$$R_X(\beta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & C\mathrm{os}\,\beta & S\mathrm{in}\,\beta \\ 0 & -S\mathrm{in}\,\beta & C\mathrm{os}\,\beta \end{pmatrix}$$

$$R_Y(\gamma) = \begin{pmatrix} C\mathrm{os}\,\gamma & 0 & -S\mathrm{in}\,\gamma \\ 0 & 1 & 0 \\ S\mathrm{in}\,\gamma & 0 & C\mathrm{os}\,\gamma \end{pmatrix}$$

Перемножив эти матрицы, получим матрицу R поворота последовательно вокруг осей $Z_\alpha X_\beta Y_\gamma$:

$$R = R_Z R_X R_Y =$$
$$= \begin{pmatrix} c_\alpha c_\gamma - s_\alpha s_\beta s_\gamma & -c_\beta s_\alpha & c_\alpha s_\gamma + c_\gamma s_\alpha s_\beta \\ c_\gamma s_\alpha + c_\alpha s_\beta s_\gamma & c_\alpha c_\beta & s_\alpha s_\gamma - c_\alpha c_\gamma s_\beta \\ -c_\beta s_\gamma & s_\beta & c_\beta c_\gamma \end{pmatrix},$$

где c обозначает косинус, а s – синус соответствующего угла.

При вычислении углов поворота из этой матрицы возникает проблема в случае, когда угол β поворота вокруг оси X равен ±90 градусам. Эта ситуация носит название «блокировки кардана». [12]. В случае, когда β=90° получаем $\mathrm{Cos}\,\beta = 0,\ \mathrm{Sin}\,\beta = 1$ и матрица поворота будет выглядеть следующим образом:

$$R_{\beta=90^\circ} = \begin{pmatrix} \cos(\alpha+\gamma) & 0 & \sin(\alpha+\gamma) \\ \sin(\alpha+\gamma) & 0 & -\cos(\alpha+\gamma) \\ 0 & 1 & 0 \end{pmatrix}$$

Видно, что матрица зависит от суммы углов, поэтому невозможно однозначно вычислить каждый из них. В этом случае один из углов (например, α) можно взять равным нулю, а второй угол вычислить. Мы рассмотрим отдельно три случая: когда угол β близок к 90°, когда он близок к (-90°) и когда угол принимает другие значения. Близость угла к ±90° можно проверить с помощью достаточно малого значения ε. Элементы матрицы R будем указывать с помощью индексов: первый индекс задает номер строки (от 1 до 3), второй – номер столбца (также от 1 до 3). Мы будем использовать также функцию atan2(x, y), см. [13] Таким образом, алгоритм будет иметь следующий вид:

Начало.

Если (1 - R_{32} < ε), то // угол β близок к 90°,

α = 0;

β = π/2;

γ = atan2(R_{21}, R_{11});

иначе, если (1 + R_{32} < ε), то

// угол β близок к (-90°)

α = 0;

β = -π/2;

γ = atan2($-R_{21}, R_{11}$);

иначе

α = atan2($-R_{12}, R_{22}$);

β = asinf(R_{32});

γ = atan2($-R_{31}, R_{33}$);

Конец.

6. Заключение

В данной работе рассматривается подключение, настройка и технология работы с устройством HTC VIVE Tracker, а также предложен алгоритм вычисления углов Тейта-Брайана ориентации трекера из матрицы перехода, получаемой с помощью библиотеки Open VR. При этом разрешается проблема «блокировки кардана».

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН «Проведение фундаментальных научных исследований (47 ГП)» по теме № FNEF-2022-0012 «Системы виртуального окружения: технологии, методы и алгоритмы математического моделирования и визуализации».

Calculation of Tait–Bryan Angles of HTC VIVE Tracker Orientation

I.P. Sablin, M.V. Mikhaylyuk, D.V. Omelchenko, D.A. Kononov, D.M. Loginov

Abstract. The paper considers the connection, configuration and technology of working with the HTC Vive Tracker 2.0. A method is also proposed for translating the transition matrix received from the tracker into Tait–Bryan angles, taking into account the "gimbal lock" problem.

Keywords: Virtual environment system, HTC Vive Tracker, Steam VR, OpenVR API, Tait–Bryan angles

Литература

1. Polona Caserman, Augusto Garcia-Agundez, Robert Konrad, Stefan Göbel, Ralf Steinmetz. Real-time body tracking in virtual reality using a Vive tracker. Virtual Reality vol. 23, p. 155–168 (2019). https://link.springer.com/article/10.1007/s10055-018-0374-z (дата обращения: 25.11.2021).

2. Yong Wang, Peng Tian, Yu Zhou, Mao Mao Zhu, Qing Chen and Chang Zhang. Using VIVE Tracker to Detect the Trajectory of Mobile Robots. EasyChair Preprint № 6241, August 5, 2021. https://www.easychair.org/publications/preprint_open/bt8n (дата обращения: 25.11.2021).

3. Susanne M. van der Veen, James S. Thomas. A Pilot Study Quantifying Center of Mass Trajectory

during Dynamic Balance Tasks Using an HTC Vive Tracker Fixed to the Pelvis. Sensors, vol. 21, (2021). https://www.mdpi.com/1424-8220/21/23/8034 (дата обращения: 25.11.2021).

4. Magdalena Żuk, Magdalena Wojtków, Michał Popek, Jakub Mazur, Katarzyna Bulińska. Three-dimensional gait analysis using a virtual reality tracking system. Measurement, vol. 188 (2022). https://www.sciencedirect.com/science/article/pii/S0263224121014974 (дата обращения: 25.11.2021).

5. Tamara von Sawitzky, Thomas Grauschopf, Andreas Riener. The Next Stage of Road Traffic Education: A Mixed Reality Bicycle Simulator to Improve Cyclist Safety. https://dl.gi.de/bitstream/handle/20.500.12116/33434/GI_VRAR_20_paper_31.pdf?sequence=1&isAllowed=y (дата обращения: 30.11.2021).

6. Lighthouse tracking. https://shazoo.ru/2015/05/18/30236/kak-rabotaet-sistema-otslezhivaniya-polozheniya-v-prostranstve-ot-valve-lighthouse (дата обращения: 30.11.2021).

7. Lighthouse tracking. https://habr.com/ru/post/369553/ (дата обращения: 30.11.2021).

8. Инструкция по установке базовых станций. https://www.vive.com/eu/support/vive-pro/category_howto/tips-for-setting-up-the-base-stations.html (дата обращения: 30.11.2021).

9. Документация OpenVR. https://partner.steamgames.com/doc/features/steamvr/openvr (дата обращения: 30.11.2021).

10. GitHub OpenVR. https://github.com/ValveSoftware/openvr (дата обращения: 30.11.2021)

11. Углы Эйлера. https://en.wikipedia.org/wiki/Euler_angles (дата обращения: 30.11.2021)

12. Блокировка кардана. https://ru.wikipedia.org/wiki/Складывание_рамок (дата обращения: 30.11.2021)

13. Функция atan2. https://e\n.wikipedia.org/wiki/Atan2 (дата обращения: 30.11.2021)

Планирование работ с неопределенными длительностями в системах реального времени

М.Г. Фуругян

ФИЦ ИУ РАН, Москва, Россия, rtsccas@yandex.ru

Аннотация. Рассматривается задача планирования работ, выполняемых на многопроцессорной системе. Работы характеризуются директивными интервалами и неопределенными длительностями, которые могут принимать значения из заданных интервалов, образующих многомерный параллелепипед. Предлагается алгоритм разбиения этого параллелепипеда на подмножества, для каждого из которых допустимое расписание имеет неизменную структуру.

Ключевые слова: многопроцессорная система, неопределенные длительности, допустимое расписание, директивный интервал, многомерный параллелепипед

1. Введение

Одно из неотъемлемых требований, предъявляемых к вычислительным системам реального времени, заключается в том, что каждая работа должна выполняться в своем директивном интервале, т.е. начаться не ранее и завершиться не позднее строго определенных моментов времени. Поэтому при разработке и сопровождении таких систем возникают различные задачи планирования работ и составления расписаний, удовлетворяющих определенным временным критериям. По данной тематике имеется большое число публикаций. Так, например, в [1, 2] исследованы различные классические задачи планирования работ и составления расписаний (обслуживание с заданными директивными сроками, задачи на быстродействие, на минимизацию суммарного и максимального временного смещения). В [3–5] задачи составления расписаний в вычислительных системах реального времени исследованы с помощью конечных автоматов с остановкой таймера и временных диаграмм. В [6] предлагаются точные и приближенные алгоритмы решения некоторых NP-трудных задач на быстродействие и минимизацию максимального временного смещения для одного и нескольких приборов. В [7, 8] рассмотрены задачи составления многопроцессорных расписаний без прерываний и переключений с нефиксированными длительностями. Положив в основу исследований метод “ветвей и границ”, авторы разработали алгоритм построения многогранников устойчивости расписаний, в каждом из которых расписание имеет неизменную структуру. Это позволяет существенно сократить вычисления, выполняемые в реальном масштабе времени. В [9, 10] некоторые задачи планирования работ сведены к минимаксным задачам. В [11] предложен псевдополиномиальный алгоритм решения задачи построения оптимального по быстродействию расписания исполнения заданий с логическими условиями предшествования. В этой задаче для каждого задания дан список его непосредственных предшественников, а также число завершенных непосредственных предшественников, необходимое для начала его выполнения. Задача сведена к циклической игре. В [12– 14] предполагается, что для выполнения заданий используется неоднородный комплекс ресурсов – возобновляемых и невозобновляемых. Для построения допустимых расписаний с прерываниями разработана сетевая потоковая модель. Решение исходной задачи сведено к поиску потока в этой модели.

В настоящей статье рассматривается задача эффективного планирования работ (заданий) в многопроцессорной системе в реальном масштабе времени. Предполагается, что в ходе проведения некоторого эксперимента необходимо выполнить определенную совокупность заданий. Каждое задание характеризуется фиксированным директивным интервалом и нефиксированной длительностью, которая может принимать целые значения из заданного временного интервала. Совокупность таких интервалов образует многомерный параллелепипед, каждая точка которого определяет некоторый вектор длительностей выполнения работ, который становится известным только в ходе проведения эксперимента, т.е. в реальном масштабе времени. Поэтому предлагается проводить расчет расписаний заранее, т.е. до проведения эксперимента, для всевозможных значений длительностей заданий. Однако, при большом числе работ сделать это практически невозможно, поскольку

число различных векторов длительностей растет экспоненциально с ростом числа заданий. Задача заключается в том, чтобы разбить указанный выше параллелепипед на области, внутри каждой из которых расписание не изменяет своей структуры. Это позволит для каждой такой области вычислять расписание только в какой-нибудь одной точке. В то же время, число таких областей должно быть существенно меньше числа целочисленных точек параллелепипеда.

2. Постановка задачи

Комплекс работ (заданий) $W = \{1, 2, \dots, n\}$, занумерованных от 1 до n, выполняется на m-процессорной системе. Каждое задание $i \in W$ характеризуется директивным интервалом $[c_i, d_i]$ и длительностью t_i, которая может принимать натуральные значения из интервала $[t_i^1, t_i^2], t_i^1, t_i^2 \in N, i = \overline{1,n}$, N–множество натуральных чисел. Пусть $N_i = \{t_i^1, t_i^1 + 1, \dots, t_i^2\}$ – возможные значения длительностей задания $i \in W$, $\Omega_0 = N_1 \times N_2 \times \dots \times N_n$. Предполагается, что проводится некоторый эксперимент (например, испытание самолета), в ходе которого вычисляются некоторые параметры, от которых зависят длительности выполнения заданий. Таким образом, характеристики работ становятся известными только в ходе проведения эксперимента, т.е. в режиме реального времени. Основная задача, которую требуется решить – это составить допустимое расписание выполнения заданий W (т.е. такое расписание, при котором каждая работа $i \in W$ выполняется строго в своем директивном интервале $[c_i, d_i]$) или определить, что такого расписания не существует. Есть несколько путей решения этой задачи. Первый – строить расписание в режиме реального времени, т.е. сразу после того, как станут известными длительности выполнения работ. Однако, это может привести к существенным временным задержкам при проведении эксперимента. Второй подход заключается в предварительном (до проведения эксперимента) построении расписания для каждого целочисленного временного вектор $(\tau_1, \tau_2, \dots, \tau_n) \in \Omega_0$. Однако, число таких векторов, равное $\prod_{i=1}^{n} T_i$, $(T_i = t_i^2 - t_i^1)$, может быть очень велико, что делает невозможным и этот подход. Мы предлагаем разбить множество Ω_0 на подмножества (число которых существенно меньше числа точек в Ω_0), такие, что в пределах каждого такого подмножества структура допустимого расписания остается неизменной. В этом случае его можно будет вычислять до проведения эксперимента для одного временного вектора из каждого подмножества.

Для фиксированного временного вектора $(\tau_1, \tau_2, \dots, \tau_n) \in \Omega_0$ допустимое расписание может быть найдено с помощью известных алгоритмов. Например, с помощью псевдополиномиального алгоритма в случае, когда работы не допускают прерываний и переключений с одного процессора на другой, а число процессоров фиксировано [15]. Для случая, когда работы допускают прерывания и переключения, может быть использован полиномиальный алгоритм, основанный на построении сетевой модели и поиске максимального потока в ней [1, 16]. Вычислительная сложность указанных алгоритмов составляет $O(nT^m)$ и $O(pn^3)$ соответственно, где $T = \max_{i=\overline{1,n}} T_i$, m–число процессоров, p–число различных типов процессоров.

3. Разбиение множества Ω_0 на подмножества

Независимо от того, какая именно задача составления расписания рассматривается, для $\tau \in \Omega_0$ введем обозначения: $f(\tau) = 0$, если допустимого расписания не существует, и $f(\tau) = 1$ в противном случае, и пусть $R(\tau)$–допустимое расписание, построенное для временного вектора τ. Иными словами, предполагается, что имеется некоторая программа $P(\tau)$, позволяющая для каждого временного вектора $\tau \in \Omega_0$ решать задачу поиска допустимого многопроцессорного расписания с директивными интервалами.

У т в е р ж д е н и е 1. Если $(\tau_1, \tau_2, \dots, \tau_n) \in \Omega_0$ и $f(\tau_1, \tau_2, \dots, \tau_n) = 0$, то $f(a_1, a_2, \dots, a_n) = 0$ для $(a_1, a_2, \dots, a_n) \in \Omega_0$, где $a_j \geq \tau_j, j = \overline{1,n}$.

Д о к а з а т е л ь с т в о. Действительно, если для некоторого временного вектора $\tau = (\tau_1, \tau_2, \dots, \tau_n) \in \Omega_0$ допустимого расписания не существует, то и для вектора, компоненты которого не меньше компонент вектора τ, его также не существует.

У т в е р ж д е н и е 2. Если $(\tau_1, \tau_2, \dots, \tau_n) \in \Omega_0$ и $f(\tau_1, \tau_2, \dots, \tau_n) = 1$, то $f(b_1, b_2, \dots, b_n) = 1$ для $(b_1, b_2, \dots, b_n) \in \Omega_0$, где $b_j \leq \tau_j, j = \overline{1,n}$.

Д о к а з а т е л ь с т в о. Действительно, если для некоторого временного вектора $\tau = (\tau_1, \tau_2, \dots, \tau_n) \in \Omega_0$ допустимое расписание существует, то и для вектора, компоненты которого не превосходят компонент вектора τ, оно также существует. Отметим также, что если в расписании для каждого задания указывать процессоры, которые его выполняют, и соответствующие временные интервалы, то для вектора $(b_1, b_2, \dots, b_n)$ можно использовать то же расписание, которое построено для вектора $(\tau_1, \tau_2, \dots, \tau_n)$, т.е. расписание с той же структурой. Отличие заключается только в том, что в расписании $R(b_1, b_2, \dots, b_n)$

некоторые работы могут завершаться раньше, чем в расписании $R(\tau_1, \tau_2, \dots, \tau_n)$.

Пусть τ_i^0– ближайшее целое к величине $(t_i^2 - t_i^1)/2, i = \overline{1,n}$. Фиксируем произвольное $k, 1 \le k \le n$, и рассмотрим функцию $f(\tau_1^0, \dots, \tau_{k-1}^0 \tau_k, \tau_{k+1}^0, \dots, \tau_n^0)$, $\tau_k \epsilon [t_k^1, t_k^2] \cap N$, как функцию одной переменной τ_k. С помощью дихотомической процедуры деления отрезка $[t_k^1, t_k^2]$ пополам найдем $\tau_k^0 \in [t_k^1, t_k^2] \cap N$, такое, что $f(\tau_1^0, \dots, \tau_{k-1}^0 \tau_k^0, \tau_{k+1}^0, \dots, \tau_n^0) = 1$, $f(\tau_1^0, \dots, \tau_{k-1}^0 \tau_k^0 + 1, \tau_{k+1}^0, \dots, \tau_n^0) = 0$. Отметим, что при этом число обращений к программе $P(\tau)$ составляет $O(\log T)$.

В результате, в силу утверждения 2 получим, что

$$f(\tau_1, \tau_2, \dots, \tau_n) = 1 \quad (1)$$

при $\tau_j \le \tau_j^0$, $j = \overline{1,n}$, и для временного вектора $(\tau_1, \tau_2, \dots, \tau_n)$ можно использовать расписание, построенное для вектора $(\tau_1^0, \tau_2^0, \dots, \tau_n^0)$. Кроме того, в силу утверждения 1

$$f(\overline{\tau}_1, \overline{\tau}_2, \dots, \overline{\tau}_n) = 0 \quad (2)$$

при $\overline{\tau}_j \ge \tau_j^0, j = \overline{1,n}, j \ne k; \overline{\tau}_k \ge \tau_k^0 + 1$.

Из (1), (2) следует, что множество временных векторов Ω_0 разбилось на четыре подмножества: $\Omega_1^1, \Omega_1^2, \overline{\Omega}_1^1, \overline{\Omega}_1^2$ со следующими характеристиками.

1) Для множеств Ω_1^1 и Ω_1^2 вопрос о существовании допустимого расписания остается открытым.

2) $f(\tau) = 1$ при $\tau \in \overline{\Omega}_1^1$, т.е. для $\tau \in \overline{\Omega}_1^1$ допустимое расписание существует и имеет такую же структуру, как и для вектора $(\tau_1^0, \dots, \tau_k^0, \dots, \tau_n^0)$.

3) $f(\tau) = 0$ при $\tau \in \overline{\Omega}_1^2$, т.е. для $\tau \in \overline{\Omega}_1^2$ допустимого расписания не существует.

Таким образом, в результате указанных выше действий для половины временных векторов из Ω_0 (т.е. для $\tau \in \overline{\Omega}_1^1 \cup \overline{\Omega}_1^2$) вопрос о существовании допустимого расписания и его построении в случае положительного ответа будет решен. Вопрос остается открытым для двух множеств – Ω_1^1 и Ω_1^2, суммарное количество целочисленных векторов в которых вдвое меньше числа векторов в Ω_0. Далее, каждое из множеств Ω_1^1 и Ω_1^2 делим на два подмножества аналогично тому, как это делалось для Ω_0. В результате, Ω_1^1 разобьется на четыре подмножества $\Omega_2^1, \Omega_2^2, \overline{\Omega}_2^1, \overline{\Omega}_2^2$, такие, что $f(\tau) = 1$ при $\tau \in \overline{\Omega}_2^1$, $f(\tau) = 0$ при $\tau \in \overline{\Omega}_2^2$, т.е. вопрос о существовании и построении допустимого расписания для подмножеств $\overline{\Omega}_2^1$ и $\overline{\Omega}_2^2$ будет решен, а для Ω_2^1 и Ω_2^2 останется открытым.

Аналогично, Ω_1^2 также разобьется на четыре подмножества $\Omega_2^3, \Omega_2^4, \overline{\Omega}_2^3, \overline{\Omega}_2^4$, такие, что для $\overline{\Omega}_2^3$ и $\overline{\Omega}_2^4$ вопрос о существовании и построении допустимого расписания будет решен, а для Ω_2^3 и Ω_2^4останется открытым. При этом $|\Omega_2^1| + |\Omega_2^2| + |\Omega_2^3| + |\Omega_2^4| = 0.5(|\Omega_1^1| + |\Omega_1^2|)$, т.е. число векторов из Ω_0, для которых вопрос о существовании допустимого расписания остается открытым, после каждой операции деления на подмножества сокращается вдвое. Продолжим этот процесс для подмножеств $\Omega_2^1, \Omega_2^2, \Omega_2^3, \Omega_2^4$ и далее для вновь образовавшихся подмножеств, для которых вопрос о существовании допустимого расписания остался открытым. Указанный процесс будет завершен, когда для каждого $\tau \in \Omega_0$ вопрос о существовании и построении допустимого расписания будет решен.

Таким образом, на каждом шаге указанного выше процесса деления множества Ω_0 на подмножества число векторов $\tau \in \Omega_0$, для которых вопрос о существовании допустимого расписания остается открытым, сокращается вдвое, а число подмножеств, содержащих эти вектора, увеличивается вдвое. Для каждого такого подмножества вычислительная сложность дихотомической процедуры составляет $O(\log T)$.

Будем предполагать, что все вектора $\tau \in \Omega_0$ равновероятны. Пусть S – это вычислительная сложность программы $P(\tau)$ построения допустимого расписания $R(\tau)$. Если фиксировать некоторое $r \in N$ и остановиться после выполнения r шагов процедуры деления множества Ω_0 на подмножества (включая расчет допустимого расписания, когда оно существует), то вычислительная сложность такой процедуры составит $O(2^r S \log T)$. При этом вероятность того, что до проведения эксперимента расписание не было построено и его необходимо будет находить в реальном масштабе временив, т.е. в ходе проведения эксперимента, составляет $1/2^r$. В то же время, вычислительная сложность процедуры построения допустимого расписания для всех векторов $\tau \in \Omega_0$ составляет$O(ST^n)$. Учитывая, что T и n могут достигать больших значений, можно подобрать такое r_0, при котором $2^{r_0} S \log T \ll ST^n$, а величина $1/2^{r_0}$ достаточно мала. Иными словами, при $r = r_0$ вычислительная сложность предложенной процедуры много меньше вычислительной сложности полного перебора векторов $\tau \in \Omega_0$. В то же время, вероятность того, что при проведении эксперимента возникнет необходимость в расчете допустимого расписания $R(\tau)$, достаточно мала.

4. Заключение

Исследована задача построения допустимого многопроцессорного расписания для совокупно-

сти работ с заданными директивными интервалами и нефиксированными длительностями. Разработан метод предварительного (до проведения эксперимента в реальном масштабе времени) расчета расписаний, вычислительная сложность которого существенно меньше вычислительной сложности полного перебора всех возможных векторов длительностей. При этом существует небольшая вероятность того, что возникнет необходимость в расчете допустимого расписания в реальном масштабе времени.

Work Planning with Indefinite Duration in Real Time Systems

Meran Furugyan

Abstract. The problem of scheduling work performed on a multiprocessor system is considered. Jobs are characterized by directive intervals and indefinite durations, which can take values from given intervals that form a multidimensional parallelepiped. An algorithm is proposed for partitioning this parallelepiped into subsets, for each of which the admissible schedule has an invariable structure.

Keywords: multiprocessor system, indefinite durations, admissible schedule, directive interval, multidimensional parallelepiped

Литература

1. Танаев В.С., Гордон В.С., Шафранский Я.М. Теория расписаний. Одностадийные системы. М.: Наука, 1984, 383 с.

2. P. Brucker. Scheduling Algorithms. Heidelberg: Springer, 2007, 378 с.

3.А.Б. Глонина, В.В. Балашов. О корректности моделирования модульных вычислительных систем реального времени с помощью сетей временных автоматов.«Моделирование и анализ информационных систем», Т. 25(2018), № 2, 174 – 192.

4.А.Б. Глонина. Обобщенная модель функционирования модульных вычислительных систем реального времени для проверки допустимости конфигураций таких систем. «Вестник ЮУрГУ. Сер. Вычисл. математика и информатика», Т. 6(2017), № 4, 43 – 59.

5. А.Б. Глонина. Инструментальная система проверки выполнения ограничений реального времени для конфигураций модульных вычислительных систем.// Вестн. МГУ. Сер. 15. Вычисл. математика и кибернетика. (2020), № 3, 16 – 29.

6. А.А. Лазарев. Теория расписаний. Оценка абсолютной погрешности и схема приближенного решения задач теории расписаний. М.: МФТИ, 2008, 222 с.

7. М.А.Горский, А.В. Мищенко, Л.Г. Нестерович, М.А. Халиков. Некоторые модификации целочисленных оптимизационных задач с учетом неопределенности и риска // Известия РАН. Теория и системы управления. (2022), № 5, 106-117.

8. А.В. Мищенко, П.С. Кошелев. Оптимизация управления работами логистического проекта в условиях неопределенности // Известия РАН. Теория и системы управления. (2021), № 4, 123-134.

9. А.А. Миронов, В.И. Цурков. Минимакс в моделях транспортного типа с интегральными ограничениями // Известия РАН. Теория и системы управления. (2003), № 4, 69 – 81.

10. А.А. Миронов, В.И. Цурков. Минимакс при нелинейных транспортных ограничениях // Доклады аладемии наук. (2001),т. 381,№ 3 305 – 308.

11. Д.В. Алифанов, И.Н. Лебедев, В.И. Цурков. Оптимизация расписаний с логическими условиями предшествования // Известия РАН. Теория и системы управления. (2009), № 6, С.88 – 93.

12. М.Г. Фуругян. Планирование вычислений в многопроцессорных АСУ реального времени с дополнительным ресурсом //Автоматика и телемеханика. (2015), №3, 144 – 150.

13. М.Г. Фуругян. Составление расписаний в многопроцессорных системах с несколькими дополнительными ресурсами // Известия РАН. Теория и системы управления. (2017), № 2, 57 – 66.

14. М.Г. Фуругян. Планирование вычислений в многопроцессорных системах с несколькими типами дополнительных ресурсов и произвольными процессорами//Вестн. МГУ. Сер. 15. Вычисл. математика и кибернетика. (2017), № 3, 38 – 45.

15. М.Г. Фуругян Некоторые алгоритмы решения минимаксной задачи составления многопроцессорного расписания // Известия РАН. Теория и системы управления. (2014), №2, 50 - 56.

16. A. Federgruen, H. Groenevel. Preemptive Scheduling of Uniform Machines by Ordinary Network Flow Technique // Management Science. (1986). Vol. 32, № 3, 341 – 349.

Модель автономных агентов с основными биологическими потребностями и мотивациями

З.Б. Сохова[1], В.Г. Редько[2]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, zarema.sokhova@gmail.com;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, vgredko@gmail.com

Аннотация. В данной работе построена и исследована модель автономных агентов с несколькими основными биологическими потребностями. Каждой потребности соответствует определенная мотивация, являющаяся основой целенаправленного поведения. Агенты в популяции имеют четыре потребности: 1) безопасности, 2) питания, 3) размножения и 4) исследования. Интенсивности этих потребностей в модели выражаются числами из промежутка [0, 1] и формируют генотип агента. В работе анализируется вопрос о том, какой генотип окажется более устойчивым, исследуется роль потребностей и мотиваций.

Ключевые слова: автономные агенты, биологические потребности, мотивации

1. Введение

Важнейшим механизмом, который в значительной степени влияет на целенаправленное поведение живых организмов, являются *потребности*. В литературе встречаются следующие виды потребностей: физиологические потребности, потребности в безопасности или ухода от опасности, социальные потребности, потребности в уважении, признании, духовные потребности и др. Первостепенное значение имеют физиологические потребности, так как от них зависит возможность существования живого организма. Подходы к классификации потребностей можно условно разделить на два типа: психологические и физиологические. Физиологическая классификация исходит из того, что потребность может существовать, только если имеется конкретная нервная структура, которая отвечает за эту потребность [1]. Известная физиологическая классификация предложена П.В. Симоновым [2]. Симонов выделял три группы потребностей: 1) витальные («жизненно необходимые»), 2) зоосоциальные («внутривидовое взаимодействие»), 3) саморазвития («направлены в будущее»). Главным центром биологических потребностей в мозге считается гипоталамус и часть структур, относящихся к базальным ганглиям [1].

Возникновение потребностей у живых организмов обусловлено изменениями, которые происходят во внутренней и внешней среде организма. В каждый момент времени у организма могут быть активны разные потребности. При этом одна из потребностей может стать ведущей. Тогда у живого организма возникает мотивация – «побудительная причина, толчок к целенаправленному поведению» [3]. Эта мотивация направляет живой организм на удовлетворение исходной потребности [4].

Для исследования поведения живых организмов в вычислительной науке часто используются *автономные агенты*. С помощью них можно моделировать как биологические организмы, так и их упрощенные искусственные аналоги [5, 6]. Агент представляет собой автономную сущность, характеризующуюся следующими свойствами: автономность, неоднородность, ограниченная рациональность, расположение в пространстве, возможность обучаться, антропоморфность, наличие конкретных целей, реактивность, социальность, наличие некоторого ресурса (энергии), память [7-10].

Отметим некоторые работы, в которых проводилось моделирование автономных агентов с естественными потребностями.

В работах [11, 12] авторами предложена модель агента, имеющего четыре биологические потребности (*голод, жажду, мочеиспускание, сон*) и четыре мотивации, соответствующие каждой из этих потребностей. Агент функционирует в среде с неограниченным источником ресурсов (еда, вода). Основной целью авторов является разработка автономного агента, который обладает механизмами, позволяющими ему иметь свои собственные потребности и интересы. Основываясь на этих потребностях, агент динамически выбирает и генерирует цели, благодаря чему возможно правдоподобное поведение.

В работе [13] исследуется поведение автономных агентов с потребностями *питания, размножения* и *безопасности* и мотивациями, которые соответствуют этим потребностям. Между

потребностями вводится иерархия, вследствие чего каждый такт времени одна из потребностей и соответствующая ей мотивация являются ведущими. Агенты в модели обучаются с использованием метода обучения с подкреплением. В работе продемонстрировано формирование циклов поведения, в которых последовательно удовлетворяются потребности питания, безопасности и размножения.

Несмотря на то, что в настоящее время имеется значительное число работ, посвященных адаптивным автономным агентам, очень мало предлагается исследовательских моделей, в которых анализируются базовые биологические потребности.

В нашей работе построена и исследована модель автономных агентов с «жизненно необходимыми» биологическими потребностями: *безопасность, питание, размножение, исследование*. При этом мы рассматриваем эту модель в эволюционном контексте, не фиксируя порядок приоритетов потребностей.

Отметим, что настоящая работа развивает статью [13], а именно, дополнительно вводится поисковая потребность (потребность исследования), особое внимание уделяется деталям взаимодействия между различными потребностями и мотивациями.

2. Описание модели

Рассмотрим популяцию автономных агентов, функционирующих в клеточной среде. Число клеток равно $N_x \cdot N_y$, причем мир замкнут: если агент движется вправо на одну клетку из клетки с координатами $\{N_x, y\}$, то он попадает в клетку с координатами $\{1, y\}$, аналогично для других «границ» мира. В каждой клетке среды может находиться только один автономный агент.

В мире имеется N_F порций пищи. В начальный момент времени порции пищи случайно распределяются по клеткам мира. Если одновременно с агентом в клетке есть порция пищи, то агент может питаться. При питании агент съедает всю порцию пищи. В конце каждого такта времени в случайных клетках снова появляются недостающие до N_F порции пищи.

В любой клетке может находиться агент-хищник. При этом если в клетке с агентом находится агент-хищник, то агент «погибает». Число хищников фиксировано, оно равно N_P. Величина N_P – параметр модели. В начальный момент времени хищники случайно распределяются по клеткам мира. В конце каждого такта времени каждый хищник перемещается на одну клетку в случайном направлении.

Агент видит ситуацию в своей клетке и в четырех соседних клетках (справа, слева, сверху, снизу), а именно, агент видит, имеется ли еда, другой агент или агент-хищник в этих клетках.

Каждый агент имеет собственный энергетический ресурс R, который пополняется при питании и теряется при выполнении действий. Время t в модели дискретно.

У агента имеется четыре потребности:

1) безопасности
2) питания,
3) размножения,
4) исследования.

Будем считать, что интенсивности потребностей для каждого агента определяются величинами P_E, P_R, P_I, P_S. Эти параметры составляют генотип агента $\mathbf{G} = \{P_E, P_R, P_I, P_S\}$. Генотипы агентов задаются в начале жизни популяции (в начале расчета) случайным образом и интенсивности равномерно распределены в интервале [0, 1]. Генотип агента определяет порядок приоритетов потребностей агента. Далее будем обозначать потребности следующим образом: "1" – безопасность, "2" – питание, "3" – размножение, "4" – исследование. Всего у агента возможны 4! = 24 различных порядка приоритетов, например, такие порядки: 1,2,3,4 или 4,2,1,3, в которых наиболее приоритетны 1-я или 4-я потребности, соответственно. Одна из задач этого исследования – выяснить, какие генотипы (порядки приоритетов) окажутся более устойчивыми в изменяющейся окружающей среде.

2.1. Мотивации агентов

Каждой потребности (безопасности, питания, размножения, исследования) соответствует определенная мотивация, которая стимулирует соответствующее потребности действие. Успешность функционирования агента зависит от порядка приоритетов его потребностей. Рассмотрим подробнее, как работают мотивации у агентов.

Мотивация питания M_E активна, если уровень энергетического ресурса $R \le R_0$, где R_0 – оптимальное значение энергетического ресурса. Уровень активной мотивации M_E равен величине P_E (рис. 1*а*). Аналогично для мотивации размножения M_R. При $R \ge R_1$ мотивация размножения активна и равна P_R (рис. 1*б*), где R_1 – уровень энергетического ресурса, необходимый для размножения.

Величина мотивации, соответствующая потребности безопасности M_S, равна 0, если агент не видит хищника в своей и в ближайших 4-х клетках, и равна P_S, если агент видит хищника в одной из этих клеток (рис. 1*в*). Считаем, что автономный агент сразу погибает (съедается хищником), если он находится в одной клетке с агентом-хищником.

Основываясь на гипотезе «уменьшения неопределенности», которая предполагает, что у

многих живых организмов постоянно присутствует мотивация к исследованию среды [14], будем полагать, что мотивация исследования M_I активна в любой такт времени и равна P_I (рис. 1*г*).

Таким образом, каждый такт времени анализируются состояния внутренней и внешней среды агента и активируются определенные мотивации.

Далее агент выделяет две мотивации: 1) «**доминанту**» [15] – мотивацию, которая соответствует *ведущей* потребности, и 2) мотивацию, которая следует за доминантой. То есть, у агента каждый такт времени активны две мотивации, обозначим их *mf* и *m*s, которые определяют его поведение, где *mf* соответствует доминантой мотивации, а *ms* следующая за доминантой. Доминанта *mf* соответствует мотивации, максимальной по величине. Вторая мотивация *ms* соответствует следующей по величине мотивации. Например, если у агента порядок приоритетов мотиваций типа $\{M_R, M_E, M_S, M_I\}$, то доминанте *mf* соответствует мотивация размножения M_R, а *ms* соответствует мотивация питания M_E. Если агент не сможет выполнить действие, которое удовлетворит *доминантную мотивацию*, то он попытается выполнить действие, удовлетворяющее *вторую мотивацию*. Если и второе действие невозможно выполнить, агент выберет действие «покой».

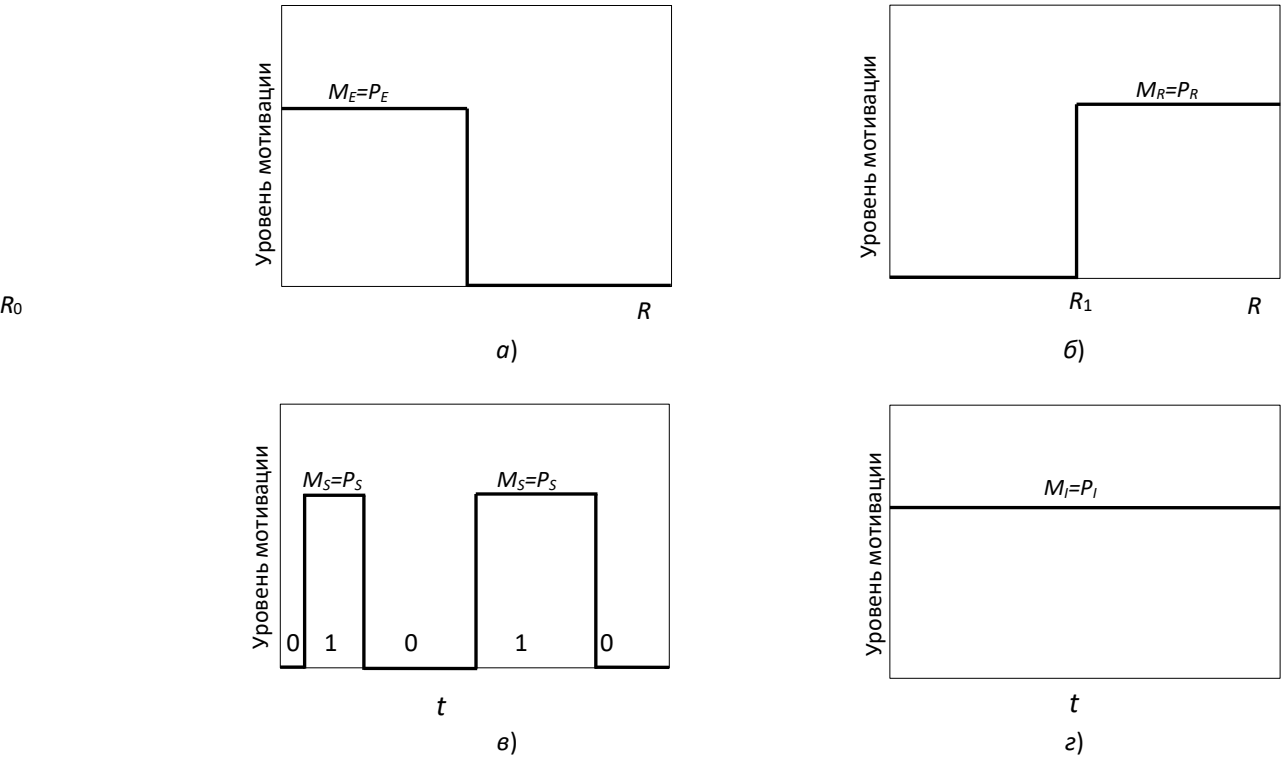


Рис. 1. Зависимости мотивации питания M_E и мотивации размножения M_R от ресурса агента R (*а*, *б*); зависимость мотивации безопасности M_S от наличия хищников в соседних ячейках (0 и 1 обозначают, есть ли на данном временном шаге хищники в соседних ячейках) (*в*) и зависимость мотивации исследования M_I от времени t (*г*)

2.2. Действия агентов

Рассмотрим подробнее возможные действия агентов.

Съедание агентов хищником. Если хищник попал в клетку с агентом, то он съедает этого агента.

Убегание от хищника. Когда агент видит хищника в соседней клетке, то у него активна мотивация безопасности M_E. Если эта мотивация становится доминантой, то агент убегает от хищника в одну из ближайших свободных клеток (где нет хищника и другого агента). Если таких свободных клеток не нашлось, то агент может погибнуть, если хищник в следующий такт времени переместится в его клетку. Наиболее предпочтительно убегание в сторону, противоположную этому хищнику. Если же нет свободных клеток, то осуществляется попытка выполнить дей-

ствие, соответствующее мотивации *ms*, в противном случае агент ничего не делает, при этом его ресурс уменьшается на небольшую величину ΔL. Величина ΔL – параметр модели.

Возможно, агенту повезет, и хищник передвинется в другую ячейку.

Питание агентов. Если агент находится в клетке с порцией пищи и доминантой является мотивация питания, то он съедает полностью эту порцию. При этом его ресурс возрастает на величину ΔR. Величина ΔR – параметр модели. Если в клетке нет пищи, то агент смотрит, есть ли пища в соседних клетках (при этом проверяется наличие в соседних клетках хищника или другого агента), если есть такая клетка, то агент передвигается в нее. Если нет, то агент смотрит на следующую мотивацию *ms*: это может быть безопасность, деление или исследование. Выполняется действие, соответствующее следующей мотивации после доминанты, если это действие возможно осуществить, иначе агент ничего не делает и теряет небольшую часть ресурса ΔL.

Размножение. Для размножения агента будем использовать репликацию – создание копии агента. Если у агента стала ведущей потребность размножения, то он реплицируется. При этом потомку передается часть ресурса агента равная qR, где $0 < q \le 0.5$. Генотип потомка $\mathbf{G} = \{P_E, P_R, P_I, P_S\}$ равен генотипу родителя с точностью до небольших мутаций (каждая из величин P_E, P_R, P_I, P_S немного варьируется, к ней добавляется случайная величина, равномерно распределенная в интервале $[-P_M, +P_M]$). Величина P_M – также параметр модели. В принципе, можно вводить разные интенсивности мутаций для разных параметров, но в данной версии модели считаем, что эти интенсивности мутаций для всех параметров одинаковы. Агент-потомок помещается в случайную свободную клетку из ближайших 4-х клеток (если в них нет другого агента). Если такой свободной клетки нет, то новый агент не рождается и происходит переход к потребности, следующей по величине мотивации *ms*. Если возможно осуществить действие, соответствующее мотивации *ms*, то агент выполняет его, иначе ничего не делает.

Поисковая активность. Если ведущей стала поисковая мотивация, то агент перемещается в одну из ближайших 4-х свободных клеток (если в ней нет другого агента или агента-хищника) в случайном направлении. Если такой свободной клетки в ближайшем окружении нет, то происходит переход к следующей мотивации *ms*. Следующие варианты могут быть питание, безопасность и размножение. Если исследование невозможно, значит, нет свободных клеток рядом и удовлетворить мотивации безопасности и размножения тоже не получится. Остается питание. Если у агента следующая после ведущей мотивации является мотивация питания и в клетке есть пища, то агент питается. Если же следующей мотивацией является безопасность или размножение, то агент ничего не делает.

Отметим, что возможна ситуация, когда два агента могут в один такт времени выбрать одну и ту же клетку для перемещения в нее, в этом случае важен порядок выбора номера агента для выбора им действия. В описанном варианте модели номера агентов для действия выбирались по порядку их появления в популяции.

Расход ресурса на действие. После выполнения любого действия ресурс агента уменьшается на величину ΔL.

Гибель агента. Агент погибает, если его съедает хищник, а также, если его ресурс стал меньше 0.

3. Результаты компьютерных экспериментов

Для анализа модели было проведено компьютерное моделирование. Основные параметры моделирования были следующие: $N_A = 300$ – начальное число агентов в мире, $N_{times} = 1000$ – число тактов времени, $P_M = 0.05$ – параметр модели, $N_x = 30$ – число клеток по вертикали, $N_y = 30$ – число клеток по горизонтали, $N_F = 200$, 400 или 600 – число порций пищи в мире, $N_P = 100$ – число хищников в мире, $\Delta R = 1.0$ – величина прироста ресурса при питании, $\Delta L = 0.1$ – величина уменьшения ресурса при выполнении любых действий, $R_0 = 2$, $R_1 = 1$, $q = 0.5$ – коэффициент, характеризующий какую долю ресурса родитель отдает потомку. Все приведенные в данном разделе результаты экспериментов усреднены по 1000 различным расчетам. Размер мира – 900 клеток.

3.1. Влияние мотиваций на динамику числа агентов и суммарный энергетический ресурс популяции

Рассмотрим сначала, как влияют мотивации на динамику числа агентов в популяции. Результаты экспериментов зависимости числа агентов с мотивациями и без мотиваций от времени приведены на рис. 2. В модели без мотиваций ведущая мотивация выбиралась случайным образом. Видно, что при «включенных» мотивациях популяция агентов выживает в отличие от популяции с «выключенными» мотивациями.

Аналогичную динамику получаем для суммарного энергетического ресурса популяции агентов с мотивациями и популяции агентов без мотиваций (рис. 3).

Следует отметить, что при увеличении количества хищников и небольшом количестве пищи в мире (например, $N_F = 200$) популяция агентов с мотивациями тоже вымирает.

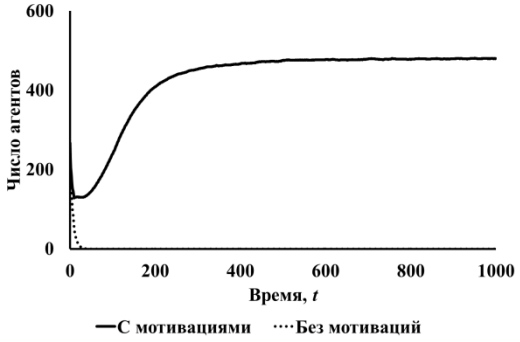


Рис. 2. Динамика численности сообщества для агентов с мотивациями и без мотиваций, $N_F = 200$

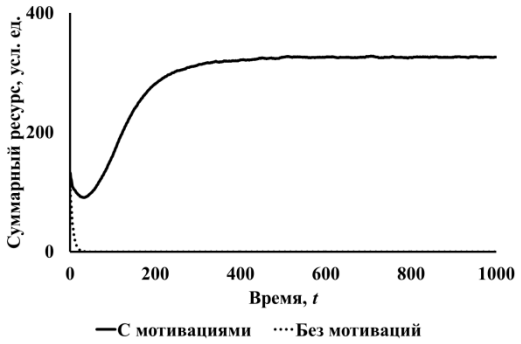


Рис. 3. Динамика суммарного энергетического ресурса сообщества агентов с мотивациями и без мотиваций, $N_F = 200$

Изменим теперь количество порций пищи в мире, пусть $N_F = 900$. Результаты для этого эксперимента представлены на рис. 4. Видно, что обе популяции выживают, но число агентов в модели с мотивациями значительно выше.

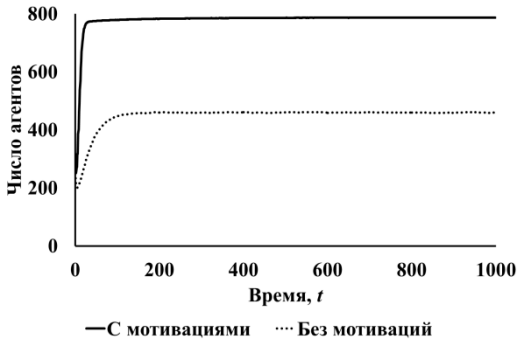


Рис. 4. Динамика суммарного энергетического ресурса сообщества агентов с мотивациями и без мотиваций, $N_F = 900$

Помимо двух представленных вариантов модели, рассмотрим модель с упрощенной схемой выбора ведущей мотивации. В этом случае агент определяет только ведущую мотивацию *mf* и не рассматривает следующую за ведущей мотивацию *ms*. Ведущая мотивация определяется следующим образом: если ресурс агента стал меньше чем R_0, то ведущая мотивация «питание»; если ресурс стал больше чем R_1, то ведущая мотивация «размножение», иначе ведущая мотивация выбирается случайно. Проведены расчеты для трех случаев: 1) с мотивациями (*mf, ms*), 2) без мотиваций, 3) упрощенная схема выбора ведущей мотивации (*mf*). Результаты (рис. 5) демонстрируют, что в модели с выбором двух приоритетных мотиваций (*mf* и *ms*) число агентов больше, чем при упрощенной схеме выбора ведущей мотивации.

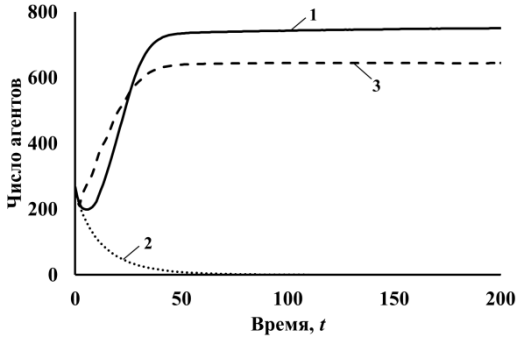


Рис. 5. Динамика численности сообщества агентов с мотивациями, без мотиваций и с упрощенной схемой выбора ведущей мотивации, $N_F = 400$ (1 – с мотивациями, 2 – без мотиваций, 3 – с упрощенной схемой выбора ведущей мотивации)

Так как популяция с выключенными мотивациями в большинстве случаев вымирает, для дальнейшего исследования остановимся на популяции с мотивациями и рассмотрим различные аспекты ее функционирования. Проанализируем сначала результаты компьютерного моделирования для различного количества пищи в мире. На рис. 6 представлены результаты экспериментов для $N_F = 200$, 400 и 600. Видно, что число агентов при $N_F = 200$ в мире значительно меньше, чем при $N_F = 400$ или 600. Отметим, что при общем числе клеток 900, максимально возможное число агентов в мире равно 800, так как остальные 100 клеток занимают агенты-хищники. Тогда получаем, что при $N_F = 400$ или 600, число агентов почти достигает предельно возможного числа.

При моделировании варьировалась также доля агентов с тем или иным генотипом в популяции. В частности, в ряде экспериментов увеличивалась доля агентов с определенным генотипом в 5 раз. Рассматривались 5 типов популяций: 0 – различные генотипы распределены по популяции равномерно (как во всех предыдущих

экспериментах). В следующих вариантах в популяции было больше в 5 раз агентов с генотипами, в которых ведущей является: 1 – потребность безопасности, 2 – потребность питания, 3 – потребность размножение, 4 – потребность исследования. При этом остальные генотипы распределены равномерно. На рис. 7 представлена динамика числа агентов в этих пяти популяциях. Видим, что в результате более успешно сообщество, в котором в начальной популяции было равномерное распределение генотипов. Отметим, что в популяциях, где было больше агентов с доминирующей потребностью исследования и безопасности сообщества прекращают свое существование (линии 1 и 4). Там, где доминирует размножение, популяция также успешно функционирует, как и популяция с равномерным распределением генотипов (линии 0 и 3). Численность агентов в популяции с доминирующей потребностью питание убывает.

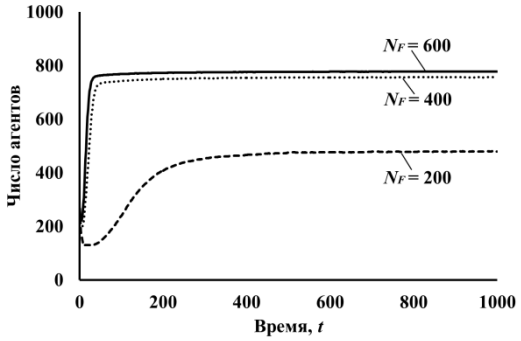


Рис. 6. Динамика численности сообщества агентов с мотивациями для различного количества пищи в мире

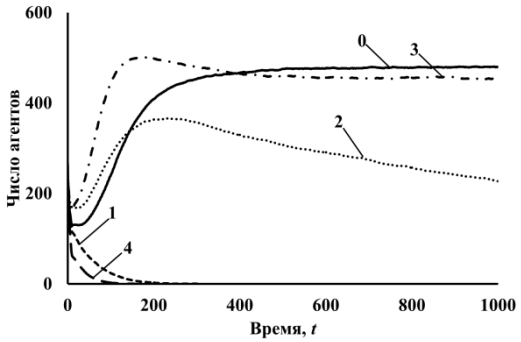


Рис. 7. Динамика численности агентов с различными долями агентов в популяции, $N_F = 200$ (0 – равномерное распределение агентов; 1 – в исходной популяции больше агентов с потребностью безопасности; 2 – в исходной популяции больше агентов с потребностью питания; 3 – в исходной популяции больше агентов с потребностью размножения; 4 – в исходной популяции больше агентов с потребностью исследования)

В дальнейших экспериментах исследовалась популяция с равномерным начальным распределением генотипов, но при этом будет показано, что в конце функционирования такого сообщества в нем будут преобладать агенты, у которых доминирует потребность размножения.

3.2. Мотивации

Рассмотрим теперь результаты ряда экспериментов, связанных с тем, как меняется средний уровень каждой из мотиваций популяции агентов в течение времени. Эксперименты проводились для разного числа порций пищи N_F в мире. В первом эксперименте число порций пищи $N_F = 200$. Видно, что в этом случае у большинства агентов на первом месте мотивация питания, так как еды в мире недостаточно. Далее следуют мотивации размножения, исследования и безопасности (рис. 8).

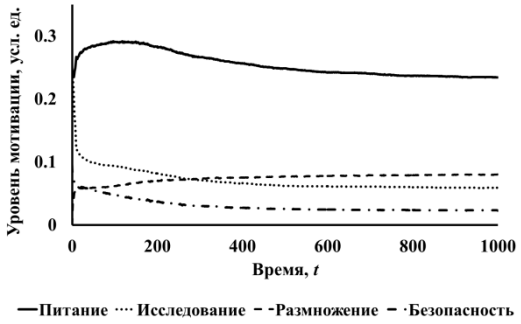


Рис. 8. Динамика среднего уровня мотиваций, $N_F = 200$

Теперь увеличим количество порций пищи в мире до $N_F = 400$. На рис. 9 представлены результаты. В этом случае на первом месте у большинства агентов мотивация размножения, затем мотивации питания, исследования и безопасности.

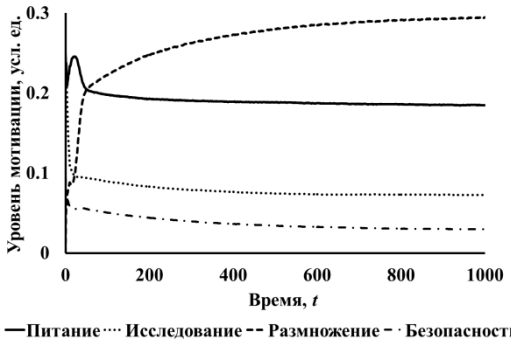


Рис. 9. Динамика среднего уровня мотиваций, $N_F = 400$

Отметим, что при проведении многочисленных экспериментов наблюдалось два основных результата: 1) у большинства агентов на первом месте мотивация питания, при малом количестве порций пищи и 2) у большинства агентов на пер-

вом месте мотивация размножения, при достаточном количестве порций пищи. Мотивации исследования и безопасности на первое место агентами не ставились.

3.3. Эволюция потребностей и генотипы

Одна из основных целей данной работы состоит в том, чтобы понять, какие генотипы из 24-х возможных окажутся более устойчивыми. При этом следует иметь в виду, что полученные результаты зависят от возможностей агентов, заложенных в модели и от начальных параметров.

Рассмотрим сначала динамику среднего уровня потребностей в популяции. Представленные ниже результаты демонстрируют как меняются эти уровни для различного числа порций пищи и агентов-хищников. На рис. 10 представлены результаты динамики среднего уровня потребностей для сообщества агентов с мотивациями при $N_F = 200$, $N_P = 100$.

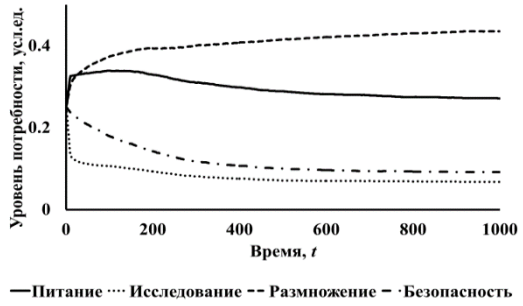


Рис. 10. Динамика среднего уровня потребностей питания, исследования, размножения и безопасности для сообщества агентов с мотивациями, $N_F = 200$, $N_P = 100$

Ниже на рис. 11 представлено распределение агентов по возможным в сообществе генотипам. Общее усредненное число агентов в популяции в этом случае равно 529. Генотип «3214» из них имеют 214 агентов, а «3241» – 192 агента, то есть эти агенты чаще реплицировались. Таким образом, выживают агенты, у которых в генотипе на первом месте стоит потребность размножения, а затем потребность питания.

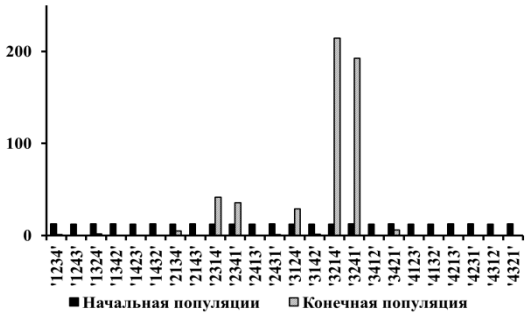


Рис. 11. Распределение агентов по генотипам в сообществе агентов с мотивациями, $N_F = 200$, $N_P = 100$

Рассмотрим теперь случай с $N_F = 600$ и $N_P = 150$, то есть увеличим число порций пищи и число агентов-хищников. На рис. 12 и 13 представлены результаты расчетов. Видим, что качественно картина почти не отличается от предыдущего случая. Отметим, что если увеличить только число хищников, не увеличивая число порций пищи, популяция агентов вымирает.

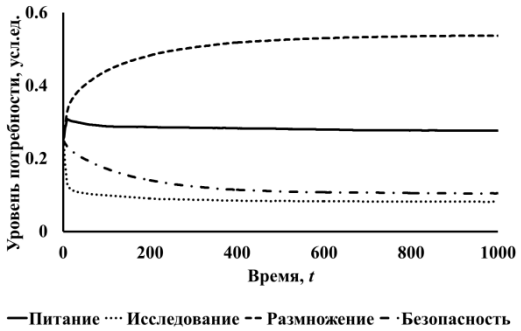


Рис. 12. Динамика среднего уровня потребностей питания, исследования, размножения и безопасности для сообщества агентов с мотивациями, $N_F = 600$, $N_P = 150$

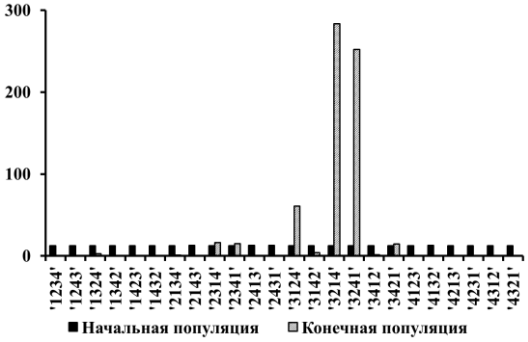


Рис. 13. Распределение агентов по генотипам в сообществе агентов с мотивациями, $N_F = 600$, $N_P = 150$

3.4. Потребность безопасности

Отметим еще один эксперимент, связанный с динамикой численности агентов. Попробуем отключить потребность безопасности. Изначально ожидалось, что в сравнении с вариантом модели, в которой потребность безопасности активна, число агентов будет ниже, но оказалось, что число агентов в популяции с отключенной потребностью безопасности выше. Результаты представлены на рис. 14. Объяснить этот результат можно следующим образом. При отсутствии потребности безопасности агенты, по-видимому, чаще реплицируются, и так как хищник движется случайно (он не видит агентов в соседних клетках), то и родитель, и потомок в некоторых случаях могут выжить оба. Возможно, если хищник будет поступать разумно (то есть, если

хищник при обнаружении агента в соседней клетке, будет в эту клетку передвигаться), то значимость безопасности возрастет. Отметим также, что в исследуемом варианте модели, в случае, когда у агентов отключена потребность безопасности, больше агентов «умирает».

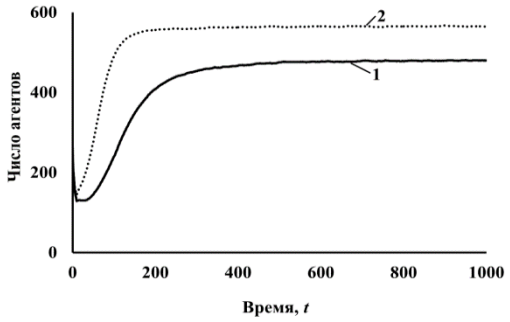


Рис. 14. Динамика численности сообщества для агентов с мотивациями, $N_F = 200$ (линия 1 – с потребностью безопасности, линия 2 – без потребности безопасности)

3.5. Эффективность агентов с потребностью исследования среды

В процессе работы над моделью возник вопрос: насколько нужна потребность исследования, так как агенты при исследовании не запоминают никакой информации. Были проведены эксперименты для модели с потребностью исследования и для модели без исследований для $N_F = 200$ и $N_F = 400$. Результаты представлены на рис. 15 и 16. Видно, что, когда в мире мало порций пищи потребность исследования оказывает существенное влияние на результаты моделирования, так как в модели, где агенты исследуют среду число агентов выше. Если же еды в мире достаточно много, разница получается несущественная.

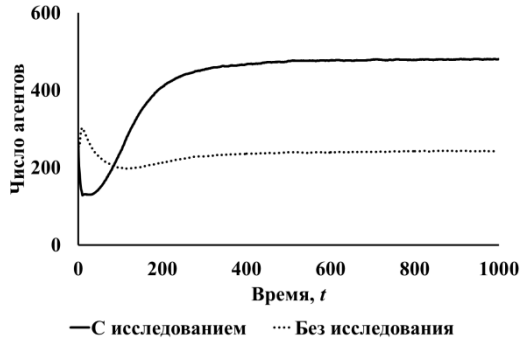


Рис. 15. Динамика численности сообщества для агентов с мотивациями. Рассматривается два случая: 1) с потребностью исследования у агентов и 2) без потребности исследования, $N_F = 200$

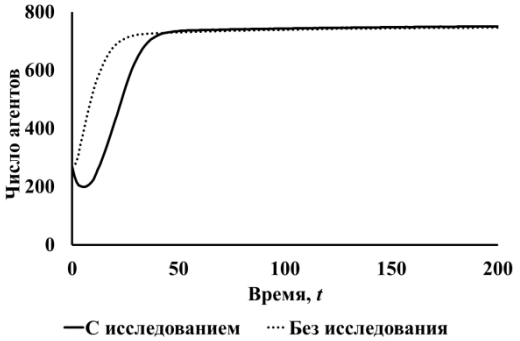


Рис. 16. Динамика численности сообщества агентов для агентов с мотивациями. Рассматривается два случая: с потребностью исследования у агентов и без потребности исследования, $N_F = 400$

4. Заключение

Таким образом, в работе изложена модель автономных агентов с основными биологическими потребностями и мотивациями. Результаты данного исследования показывают, что наиболее устойчивыми являются генотипы, в которых ведущие потребности размножение и питание. Также продемонстрирована важная роль потребности исследования при малом числе порций пищи в среде, несмотря на то, что агент при исследовании не запоминает никакой информации, но при этом имеет больший шанс найти источник энергии, если он чаще двигается по миру, как это и происходит в естественных биологических системах.

По мнению авторов, можно улучшить модель, добавив возможность для автономных агентов при исследовательском поведении строить карту местности и формировать базу знаний, чтобы потом использовать ее. При этом порции пищи и агенты-хищники должны появляться не случайным образом в мире, а в некоторых клетках мира чаще, чем в других, т.е. нужно внести какую-либо закономерность, которую агент смог бы выучить, функционируя в среде. Как было отмечено в разделе 3.4 стоит также добавить для агентов-хищников возможность действовать более разумно.

Настоящая работа выполнена в рамках государственного задания по проведению фундаментальных научных исследований по теме «Исследование нейроморфных систем обработки больших данных и технологии их изготовления». Проект № FNEF-2022-0003.

A Model of Autonomous Agents with Basic Biological Needs and Motivations

Zarema B. Sokhova, Vladimir G. Red'ko

Abstract. In this paper, a model of autonomous agents with several basic biological needs is constructed and investigated. Each need corresponds to a certain motivation, which is the basis of purposeful behavior. Agents in a population have four needs: 1) safety, 2) nutrition, 3) reproduction and 4) research. The intensities of these needs in the model are expressed by numbers from the interval [0, 1] and form the genotype of the agent. In a population, 24 different combinations of the order of priorities of needs are possible. The model was investigated by computer simulation. The paper analyzes the question of which genotype will be more stable, examines the role of needs and motivations.

Keywords: autonomous agents, biological needs, biological motivations

Литература

1. Н.А. Фонсова, И.Ю. Сергеев, В.А. Дубынин. Анатомия центральной нервной системы. М., ЮРАЙТ, 2016.
2. П.В. Симонов. Мотивированный мозг. Высшая нервная деятельность и естественно-научные основы общей психологии. Питер, 2023.
3. А.И. Лакомкин, И.Ф. Мягков. Биологические потребности и мотивации. Воронеж, ВГУ, 1980.
4. К.В. Судаков. Биологические мотивации. М., Медицина, 1971.
5. В.Г. Редько. Моделирование когнитивной эволюции: На пути к теории эволюционного происхождения мышления. М., ЛЕНАНД, 2020.
6. В.А. Непомнящих. Аниматы как модель поведения животных. IV Всероссийская научно-техническая конференция «Нейроинформатика-2002». Материалы дискуссии «Проблемы интеллектуального управления – общесистемные, эволюционные и нейросетевые аспекты». М., МИФИ, 2003.
7. N. Gilbert. Agent-based models. Sage Publications, Inc, 2007.
8. B. Hayes-Roth. An architecture for adaptive intelligent systems. «Artificial Intelligence: Special Issue on Agents and Interactivity», V. 72 (1995), 329–365.
9. M. Wooldridge, N. Jennings. Intelligent agent: theory and practice. «Knowledge Engineering Review», V. 10 (1995), № 2, 115−152.
10. P. Maes. Artificial Life meets entertainment: life like autonomous agents. «Communications of the ACM», V. 38 (1995), № 11, 108−114.
11. N. Avradinis, T. Panayiotopoulos, G. Anastassakis. Modelling basic needs as agent motivations. «International Journal of Computational Intelligence Studies», V. 2 (2013), № 1, 52–75.
12. N. Avradinis, T. Panayiotopoulos, G.Anastassakis. Behavior believability in virtual worlds: agents acting when they need to. «SpringerPlus», V. 2 (2013), 246.
13. А.Г. Коваль, В.Г. Редько. Поведение модельных организмов, обладающих естественными потребностями и мотивациями. «Математическая биология и биоинформатика», Т. 7 (2012), № 1, 266–273.
14. В.А. Непомнящих. Связь между автономным и адаптивным поведением у искусственных агентов и животных. Сборник научных трудов: Подходы к моделированию мышления. Под ред. В.Г. Редько. М., ЛЕНАНД, 2019.
15. А.А. Ухтомский. Доминанта. Статьи разных лет. 1887-1939. СПб., Питер, 2002.

Как автономный когнитивный агент может создавать аксиоматическую теорию

В.Г. Редько

ФГУ ФНЦ НИИСИ РАН, Москва, Россия, vgredko@gmail.com

Аннотация. Может ли компьютерный автономный агент сам «изобрести» аксиоматический метод и применить его в определенной математической теории. В настоящей статье обсуждается этот вопрос. В качестве прототипа возможной аксиоматической теории используются «Начала» Евклида.

Ключевые слова: аксиомы, постулаты, теоремы, аксиоматическая теория

1. Введение

Один из ярких научных методов – аксиоматический подход. Основа этого подхода была заложена в «Началах» Евклида (примерно 300 год до н.э.) [1]. В «Началах» используются определения, постулаты и аксиомы, на основе которых дедуктивно доказываются многочисленные предложения, а именно, задачи (в которых нужно что-то построить) и теоремы (в которых нужно что-то доказать). Хотя сборник [1] содержит 6 книг, в целом «Начала» Евклида содержат 15 книг (подробнее см., например, [2]).

Необходимо отметить, что предпосылками создания «Начал» Евклида послужили работы академии Платона, в которой лицеисты решали многочисленные математические задачи, в основном геометрические. Эта академия была организована Платоном примерно в 387 г. до н. э. близ Афин. Платон подчеркивал важность решения математических задач для развития мышления членов академии. «Начала» Евклида излагаются как раз в духе работ членов академии Платона.

До появления «Начал» Евклида труды с таким же названием, суть которых заключалась в последовательном изложении ключевых фактов теоретической арифметики и геометрии, были составлены Гиппократом Хиосским (вторая половина V века до н. э.), а также платониками Леонтом и Февдием. Все они практически исчезли из обихода после появления работы Евклида. Об этом см., например, [2].

«Начала» Евклида долгое время служили образцом логического изложения математической теории. Например, структура «Математических начал натуральной философии» И. Ньютона [3] прямо соответствовала структуре «Начал» Евклида. Таким образом, «Начала» Евклида [1] послужили мощным образцом аксиоматической теории.

Может ли компьютерный автономный агент сам «изобрести» аксиоматический метод и применить его в определенной математической теории. В настоящей статье обсуждается этот вопрос. В качестве прототипа возможной аксиоматической теории будем использовать «Начала» Евклида. Анализ начнем с модели агента-лицеиста в академии Платона.

2. Пример агента-лицеиста в академии Платона

Иллюстративная модель учёбы агента-лицеиста в академии Платона была построена в работе [4]. Воспроизведем основные результаты этой работы. Была построена простая компьютерная модель. Считалось, что агент в процессе учёбы накапливает математические знания, осваивает методы решения задач. При удачном решении задач у агента формируется уверенность в своих математических способностях, при этом возрастает и вероятность решения следующих задач. Определялась зависимость вероятности решения задачи агентом $P(t)$ от времени t. Считалось, что при $t = 0$ агент-лицеист только что поступил в академию, так что $P(0)$ мало; время t дискретно, один такт времени соответствует попытке решения одной задачи агентом. Пример расчета зависимости $P(t)$ по модели работы [4] приведен на рис. 1.

Видно, что сначала агент-лицеист довольно длительное время обучается и решает задачи редко. Затем вероятность решения задачи растет и приближается к 1. Явно видны скачки роста $P(t)$ в моменты правильного решения задач.

Понятно, что для коллектива академии Платона был важен обмен информацией о методах решения задач и о полученных результатах. Причем были важны красивые, серьезные, нетривиальные результаты, например, такие, как теорема Пифагора. Поэтому были полезны обзоры этих результатов. И такие обзоры появлялись еще до «Начал» Евклида.

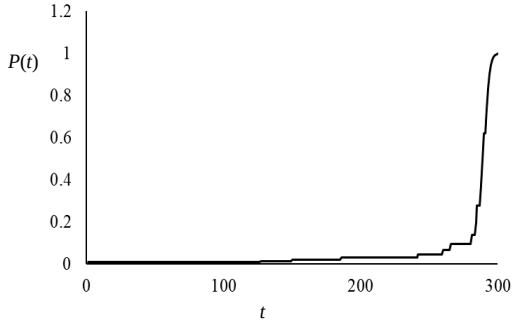


Рис. 1. Зависимость вероятности правильного решения задачи $P(t)$ агентом-лицеистом от времени t

Как же автономный агент может самостоятельно построить аксиоматическую теорию, аналогичную «Началам» Евклида? Для того чтобы попробовать ответить на этот вопрос, кратко охарактеризуем сами «Начала» Евклида.

3. Краткая характеристика содержания «Начал» Евклида

Остановимся на книге I, которая характеризует геометрические свойства плоских фигур [1]. Начинается книга с **определений** основных понятий; определений точки, линии, прямой линии, поверхности, плоской поверхности, угла, острого, тупого и прямого углов, круга, окружности, центра и диаметра круга, треугольника, равностороннего, равнобедренного и разностороннего треугольников, квадрата, параллелограмма, параллельных прямых.

За определениями следуют **постулаты**:

«1. От всякой точки до всякой точки можно провести прямую.

2. Ограниченную прямую можно непрерывно продолжать по прямой.

3. Из всякого центра и всяким раствором может быть описан круг.

4. Все прямые углы равны между собой.

5. Если прямая, падающая на две прямые, образует внутренние и по одну сторону углы, меньшие двух прямых, то продолженные эти две прямые неограниченно встретятся с той стороны, где углы меньшие двух прямых.»

За постулатами следуют **аксиомы**, которые имеют характер общих утверждений, относящихся как геометрическим понятиям, так и к другим величинам:

«1. Равные одному и тому же равны и между собой.

2. И если к равным прибавляются равные, то и целые будут равны.

3. И если от равных отнимаются равные, то остатки будут равны.

4. И если к неравным прибавляются равные, то целые будут не равны.

5. И удвоенные одного и того же равны между собой.

6. И половины одного и того же равны между собой.

7. И совмещающиеся друг с другом равны между собой.

8. И целое больше части.

9. И две прямые не содержат пространства.»

Детальнее об определениях, постулатах и аксиомах см. [1], в частности, комментарии переводчика в [1].

Отметим, что постулаты имеют характер аксиом.

Основываясь на определениях, постулатах и аксиомах, Евклид излагает и доказывает теоремы (часть из теорем являются построениями фигур). Приведём без доказательства первые пять теорем. Теоремы в [1] называются Предложениями.

Предложение 1. «На данной ограниченной прямой построить равносторонний треугольник.»

Под ограниченной прямой подразумевается отрезок определенной длины. При доказательстве этого предложения используются постулаты 1 и 3, а также аксиома 1.

Предложение 2. «От данной точки отложить прямую, равную данной прямой.»

При доказательстве этого предложения используются постулат 3, аксиома 1 и предложение 1.

Предложение 3. «Из двух заданных неравных прямых от большей отнять прямую, равную меньшей.»

При доказательстве этого предложения используются постулат 3, аксиома 1 и предложение 2.

Предложение 4. «Если два треугольника имеют по две стороны, равные каждая каждой, и по равному углу, содержащемуся между равными прямыми, то они будут иметь и основание, равное основанию, и один треугольник будет равен другому, и остальные углы, стягиваемые равными сторонами, будут равны остальным углам каждый каждому.»

Это хорошо известное свойство равенства треугольников. При доказательстве этого предложения используется, аксиома 9.

Предложение 5. «У равнобедренных треугольников углы при основании равны между собой, и при продолжении равных прямых углы под основанием будут равны между собой.»

При доказательстве этого предложения используются постулат 2, аксиома 3 и предложения 3 и 4.

Из приведенных предложений видно, что предложения постепенно становятся сложнее и

при доказательствах используются как постулаты и аксиомы, как и уже доказанные предложения.

Всего в книге I доказано 48 предложений. Заканчивается эта книга доказательством теоремы Пифагора.

4. «Основания геометрии» Гильберта

Отметим, что определенное развитие «Начал» Евклида сделал Д. Гильберт в книге «Основания геометрии» [5] (первое издание этой монографии вышло в 1899 году). Классические «Основания геометрии» Гильберта стали образцом для дальнейших работ по аксиоматическому построению геометрии. Гильберт реализовал её с исчерпывающей полнотой. Он не только дал полную аксиоматику геометрии, но также детально проанализировал эту аксиоматику, доказав независимость каждой из своих аксиом. Подчеркнём, что Гильберт построил новую систему аксиом, которые во многом близки к аксиомам Евклида, но явно от них отличаются.

5. Автоматизированные системы доказательства теорем

Как же может создавать подобную аксиоматическую теорию компьютерный автономный агент? Представим себе коллективного агента-лицеиста академии Платона. Имеется область – геометрия – в которой этот агент решает задачи. Технический метод решения задач известен: использовать только линейку и циркуль. Нужны и мысленные методы: постановка задач и использование доказательств при решении задач. Предположим, что наш агент освоил эти методы. Далее он решает различные задачи и составляет обзоры получаемых результатов. Особенно нетривиальным процессом является доказательство.

Кратко остановимся на методах создания компьютерных систем, способных самостоятельно производить доказательства. Такие системы были разработаны в 1950-годах А. Ньюэллом, Г.А. Саймоном и К. Шоу. Была разработана программа «Логик-теоретик» в рамках символической логики. Подробнее см. сайт [6], на котором представлен и отчёт А. Ньюэлла и Г.А. Саймона фирмы RAND Corporation, характеризующий суть программы и саму программу [7].

Основные особенности метода и программы «Логик-теоретик» состоят в следующем. Используется формализованная символическая логика. Имеются аксиомы, также формализованные в символической логике, хотя имеется и интерпретация этих аксиом. Имеются определения логических выражений. Используются элементарные выводы (например, методом аналогичным “modus ponens”), замены логических выражений их определениями. Доказываются теоремы в рамках формализованной символической логики. Используются списки уже доказанных теорем. Полезны списки нерешенных проблем. Для сокращения процессов развития аксиоматической теории используются определенные эвристики: рассматривается подобие символических выражений, подобие теорем. Рассматриваются процедуры обучения, при которых накапливается опыт.

Это была первая программа, специально разработанная для выполнения автоматических рассуждений. Логик-теоретик доказал некоторые теоремы в известных Principia Mathematica Б. Рассела и А.Н. Уайтхеда [8].

Таким образом, была разработана автоматизированная компьютерная информационная система, способная находить, используя эвристические методы, доказательства теорем в символической логике.

Дальнейшим развитием программы «Логик-теоретик» стала компьютерная программа «Универсальный решатель задач» или General Problem Solver (GPS), также разработанная А. Ньюэллом, Г.А. Саймоном и К. Шоу [9]. Эта программа была предназначена для работы в качестве универсальной машины для решения задач. В качестве примеров использования приводились доказательства теорем евклидовой геометрии и логики предикатов, решение шахматных задач.

Для GPS характерны следующие черты:

1) Рекурсивные процессы решения проблем
2) Отделение методов решения проблем от содержательной проблемы
3) Использование двух основных методов: а) анализ средств и целей, б) планирование
4) Память и программа организованы так, чтобы автоматизировать программу.

В этой программе активно используются эвристики. GPS работает над проблемами, сформулированными в виде объектов и операторов. Оператор преобразует одни объекты в другие. При доказательстве теорем объекты – теоремы, операторы – допустимые правила вывода. Программа – цепочка операторов.

Используются цепочки целей и подцелей. Ядро GPS состоит из некоторых общих, но достаточно мощных эвристик. Используется эвристика уменьшения подцелей. Формируется организованная система эвристик. Рассматривается сходство между объектами.

При планировании используется 1) абстрагирование от некоторых деталей объектов и операторов, 2) формирование соответствующей проблемы в абстрактном виде, 3) после решения абстрактной проблемы формирование плана оригинальной проблемы, 4) перевод плана в исходный вид и реализация его.

Имеются процедуры обучения, при которых накапливается и используется ранее полученный опыт.

Возможно разбиение большой проблемы на ряд малых. Также используется итеративный процесс с формированием новых планов.

Подробнее об универсальном решателе задач см. [9]. См. также книгу [10], развивающую работы по GPS.

Используя GPS, Г. Гелертер с коллегами разработали программы, прямо предназначенные для доказательства теорем евклидовой геометрии [11-13]. В этих программах вводятся специальные обозначения геометрических терминов, т.е. используется специальная символика. Используются специальные графы целей и подцелей, т.е. «деревья» зависимости результатов и подрезультатов. Важно, что в этих работах, так же, как и в GPS, процесс доказательств использовал ряд эвристик, т.е. в программах должна быть заложена способность самостоятельно осуществлять догадки эффективных путей доказательства. Был разработан удобный специальный язык обработки списков, компилируемый системой FORTRAN.

В работах [11-13] излагаются компьютерные доказательства нескольких геометрических теорем. В частности, в работе [12] была доказана первая теорема в «Началах» Евклида с помощью компьютерной программы, содержащей 20 000 отдельных компьютерных операторов.

Итак, в работах [11-13] были созданы компьютерные программы, осуществляющие доказательства геометрических теорем, в том числе теорем из «Начал» Евклида. Хотя в процесс доказательств включались эвристики и способность программы к самостоятельным догадкам. Сами программы были довольно сложными, что требует от автономного агента способности искусства создавать эффективные программы.

В дальнейшем появились системы по автоматическим доказательствам теорем, в которых геометрические понятия преобразуются в алгебраические равенства и неравенства. См. например, обзоры таких работ [14, 15]. То есть работы по автоматизированным доказательствам теорем активно развивались и продолжают развиваться.

В настоящее время имеется журнал “Journal of Automated Reasoning” [16], который включает тематику по развитию и применению систем автоматического доказательства теорем (automatic theorem provers).

О таких компьютерных системах см. также информацию Л.Д. Беклемишева, представленную на сайте ПостНауки [17].

6. Коллективный агент-лицеист. Шаги к аксиоматической теории

Вернёмся к коллективному агенту-лицеисту и рассмотрим, как он сможет пополнять и структурировать составленный им обзор полученных результатов, чтобы в конечном итоге получилась аксиоматическая теория, подобная «Началам» Евклида. Считаем, что обзор результатов содержит решения различных задач, а именно, задач на построение определенных фигур и доказательства теорем. Теперь рассмотрим, как коллективный агент-лицеист может преобразовать этот обзор в аксиоматическую теорию. Понятно, что теперь нужно добавить в обзор определения, т.е. охарактеризовать основные понятия, используемые при решении задач. Также необходимо добавить в обзор постулаты и аксиомы, для простоты постулаты дальше будем называть аксиомами. И самое нетривиальное, что нужно сделать, это надо упорядочить результаты обзора таким образом, чтобы полученные результаты последовательно выводились из аксиом и ранее изложенных результатов. При этом такое структурирование может потребовать введение новых понятий, аксиом и получение новых результатов, чтобы вся аксиоматическая теория представляла собой последовательно связанную цепочку элементов. По-видимому, такое структурирование будет происходить многократно, так как вполне возможно, что единая цепочка будет формироваться не сразу, а в результате многократных проверок и необходимых дополнений как определений, аксиом, так и теорем.

В начале обзора нужно поместить определения и аксиомы, стремясь к полноте изложения используемых понятий и к достаточно полной системе аксиом. Далее целесообразно излагать последовательно результаты, начиная с самых простых и наиболее общих. Простоту можно оценивать по числу используемых аксиом и по числу элементарных логических выводов, т.е. элементарных мысленных переходов вида «поскольку …, то…», «если…, то». Затем надо делать структурирование обзора, а именно, надо делать проверку получающейся системы на предмет того, что действительно каждый изложенный результат использует только систему аксиом и ранее изложенные результаты. Это структурирование осуществляется следующим образом.

Считаем, что общее число результатов, изложенных в обзоре, равно n. Вводятся номера результатов k = 1,2,…,n. Далее рассматриваем все результаты по порядку с номерами k = 1, 2, … Остановимся на случае, когда рассматривается отдельный результат с номером k^*. Рассматриваются все использованные при получении данного результата k^* ссылки на другие результаты (аналогичные ссылкам на другие предложения в предложениях «Начал» Евклида, см. выше). Если все ссылки делаются на результаты с номерами, меньшими номера данного результата k^*, то для этого результата никакой перенумерации не происходит и рассматривается следующий результат с номером k^*+1. Если имеется хотя бы одна ссылка на результат с номером, большим, чем k^*, то происходит перенос данного результата в конец списка, т.е. результату присваивается номер n+1. После этого уменьшаются на 1 все номера результатов с номерами k^*+1,…,n+1. То есть теперь номер k^* имеет результат, следовавший ранее далее за данным результатом в обзоре. Перенесенный в конец списка данный результат имеет новый номер n. В соответствии с уменьшением номеров результатов уменьшаются и номера ссылок в перенумерованных результатах, т.е. в результатах, имеющих теперь номера k^*,…,n, а именно, рассматриваются ссылки на результаты с новыми номерами. Такая процедура структурирования проводится последовательно для всех результатов с номерами k = 1,2,…,n. После проведения структурирования для всех результатов каждый из результатов будет иметь ссылки только на результаты, предыдущие в обзоре, так же, как это имеет место в «Началах» Евклида.

Снова вернёмся к коллективному агенту-лицеисту. Он может получать новые результаты, и дополнять систему аксиом. Дополнение аксиом не будет приводить к существенным изменениям в обзоре. Если новые результаты не влияют на предыдущие, то их естественно вносить в конец списка результатов. Но если новые результаты влияют на предыдущие, то это приведёт к существенному изменению списка и к необходимости нового структурирования обзора по изложенному выше методу.

7. Заключение

В настоящей работе проанализированы вопросы, связанные с построением аксиоматических теорий в компьютерных программах. В качестве прототипа возможной аксиоматической теории рассматриваются «Начала» Евклида. Проанализированы методы создания компьютерных систем, способных самостоятельно производить доказательства: программы «Логик-теоретик» и «Универсальный решатель задач» разработанные А. Ньюэллом, Г.А. Саймоном и К. Шоу в 1950-х годах. Особое внимание уделено компьютерным системам (основанным на «Универсальном решателе задач»), разработанным Г. Гелертером с коллегами (конец 1950-х годов); эти системы прямо предназначены для доказательства теорем евклидовой геометрии. Анализ этих методов показывает сложность процессов формализации геометрических понятий, при формировании программ доказательств теорем. Тем не менее, такие методы могут быть использованы достаточно эффективным компьютерным автономным агентом при создании аксиоматических теорий и соответствующих компьютерных программ. Таким образом, в настоящей работе проведён анализ, показывающий, что возможно создание компьютерного автономного агента, самостоятельно формирующего аксиоматическую теорию.

Настоящая работа выполнена в рамках государственного задания по проведению фундаментальных научных исследований по теме «Исследование нейроморфных систем обработки больших данных и технологии их изготовления», проект № FNEF-2022-0003.

How an Autonomous Cognitive Agent Can Create an Axiomatic Theory

Vladimir G. Red'ko

Abstract. Can a computer autonomous agent "invent" an axiomatic method by itself and apply it in a certain mathematical theory. This article discusses this issue. Euclid's "Elements" is used as a prototype for a possible axiomatic theory.

Keywords: axioms, postulates, theorems, axiomatic theory

Литература

1. Евклид. НАЧАЛА. Книги I–VI. (Пер. с греческого и комментарии Д.Д. Мордухай-Болтовского при редакционном участии М.Я. Выгодского и И.Н. Веселовского). М.-Л., ОГИЗ, Государственное издательство технико-теоретической литературы, 1948.
2. https://24smi.org/celebrity/4943-evklid.html
3. I. Newton. Philosophiæ Naturalis Principia Mathematica. 1687. И. Ньютон. Математические начала натуральной философии. М., Наука, 1989.
4. В.Г. Редько. Как автономный компьютерный агент может самостоятельно открывать законы природы. «Интегрированные модели и мягкие вычисления в искусственном интеллекте». Сборник научных трудов XI Международной научно-практической конференции (ИММВ-2022, Коломна, 16-19 мая 2022 г.). В 2-х томах. Т. 2. М., РАИИ, 2022, 108–118.
5. Д. Гильберт. Основания геометрии. М.-Л., ОГИЗ, Государственное издательство технико-теоретической литературы, 1948.
6. https://wiki5.ru/wiki/Logic_Theorist
7. http://shelf1.library.cmu.edu/IMLS/MindModels/logictheorymachine.pdf
8. A.N. Whitehead, B. Russell. Principia Mathematica. Cambridge, Cambridge University. Press, 2nd edition. Vol. I (XLVI + 674 p.) 1925; Vol. II (XXXI + 742 p.) 1927; Vol. III (VIII + 491 p.) 1927.
9. A. Newell, J.C. Shaw, H.A. Simon. Report on a general problem-solving program. “Proceedings of the International Conference on Information Processing”. Paris, UNESCO, 15-20 June 1959. Published in 1960 by UNESCO (Paris), R. Oldenbourg (München) and Butterworths (London), 256–264. See also: http://bitsavers.informatik.uni-stuttgart.de/pdf/rand/ipl/P-1584_Report_On_A_General_Problem-Solving_Program_Feb59.pdf
10. A. Newell. Unified Theories of Cognition. Cambridge, Massachusetts, Harvard University Press, 1990.
11. H. Gelernter, J. Hansen, D. Loveland. Empirical explorations of the geometry theorem proving machine. “Proceedings of the Western Joint Computer Conference”. 1960, Vol. 17, 143–147. Reprinted in “Computers and Thought”. E. Feigenbaum, J. Feldman (Eds.). New York, McGraw-Hill Book Co., 1963, 153–167. See also: https://dl.acm.org/doi/pdf/10.1145/1460361.1460381
12. H. Gelernter. Realization of a geometry theorem proving machine. “Proceedings of the International Conference Information Processing”, Paris, June 15-20 1959, 273–282. Reprinted in “Computers and Thought”. E. Feigenbaum, J. Feldman (Eds.). New York, McGraw-Hill Book Co., 1963, 134–152.
13. H.L. Gelernter, N. Rochester. Intelligent behavior in problem-solving machines. “IBM Journal of Research and Development”, Vol. 2 (1958), No 4, 336–345.
14. A. Ferro, G. Gallo. Automated theorem proving in elementary geometry. “Le Matematiche”. Vol. 43 (1988), No 1,2, 195–224. See also: https://lematematiche.dmi.unict.it/index.php/lematematiche/article/view/713/678
15. D.W. Loveland. Automated Theorem Proving: A Logical Basis. “Fundamental Studies in Computer Science”. Vol. 6. Amsterdam, New York, Oxford, North-Holland Publishing Company, 1978.
16. Сайт журнала “The Journal of Automated Reasoning”: https://www.springer.com/journal/10817
17. https://postnauka.ru/faq/26503

Начальное обучение алгоритмике дошкольников и других новичков с помощью умных роботов-игрушек

А.Г. Кушниренко[1], А.Г. Леонов[2], Д.В. Мащенко[3], М.В. Райко[4], И.Н. Грибанова[5]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, agk_@mail.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, dr.l@vip.niisi.ru;

[3]ФГУ ФНЦ НИИСИ РАН, г. Москва, Россия, mashchenko.darya.vlad@yandex.ru;

[4]ФГУ ФНЦ НИИСИ РАН, г. Москва, Россия, mila.rayko@gmail.com;

[5]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, nig@niisi.msk.ru

Аннотация. В современном мире алгоритмика становится в один ряд со школьной арифметикой, ведь базовые элементы программирования, освоенные дошкольником, будут сопровождать его на протяжении всей жизни в связи с глобальной цифровизацией. В статье описаны мировые практики по обучению детей алгоритмике, отмечены плюсы и минусы различных методик. Особое внимание уделено бестекстовой учебной среде программирования ПиктоМир, описан опыт организации и проведения систематических пропедевтических курсов алгоритмики и программирования для дошкольников и младшеклассников. Описана необходимость демонстрации детям обратной связи между компьютером и роботом. Авторами предложено решение, заключающееся в добавлении в среду ПиктоМир нового робота, отвечающего на команды-вопросы. Сделаны выводы о направлении дальнейших исследований, в частности, о необходимости повышения вовлеченности детей в процесс выполнения программы роботом.

Ключевые слова: алгоритмика, ПиктоМир, дошкольный курс программирования, бестекстовая среда программирования, умные роботы-игрушки, программирование с обратной связью

1. Что такое алгоритмика и почему ее должны осваивать дошкольники 21 века

Алгоритмика (азы программирования) – это универсальный набор понятий и конструкций, позволяющий людям планировать будущую деятельность, разворачивающуюся во времени. Если планируемая деятельность рассчитана на её проведение с использованием компьютеров или других автоматических устройств, то принято называть этот процесс планирования программированием. Человека, занимающегося программированием, называют программистом, а результат работы программиста называют программой. Программа материальна и может быть закодирована скоплениями электронов в ячейках памяти компьютера, или составлена из букв и цифр на бумаге или экране компьютера, или составлена педагогом из картинок-пиктограмм на магнитных карточках на классной доске или выложена ребенком из кубиков с картинками на своем рабочем столе.

Аналогично тому, как школьная арифметика является фундаментом и основанием и школьной и высшей математики, алгоритмика – набор понятий и конструкций, позволяющих описывать планы действий, разворачивающихся во времени – является фундаментом и учебной, и профессиональной информатики. Все мы знаем, что в дошкольном возрасте дети охотно осваивают на практике азы счета, оперируя материальными предметами и заучивая слова и знаки, обозначающие цифры и числа, операции сложения, вычитания и сравнения чисел. Эти знания и навыки со временем не устаревают: одними и теми же базовыми элементами арифметики, цифрами и знаками операций, выученными в раннем детстве, оперируют и дошкольники, и пенсионеры, и продавцы магазинов, и ученые ракетных наук. В алгоритмике и программировании есть свои базовые элементы: циклы, ветвления, подпрограммы, счетчики и др. Базовые навыки программирования, выученные дошкольником 21 века, пригодятся ему и в школе, и на работе, и в повседневной жизни по той причине, что на планете Земля сегодня, в первой четверти 21 века, завершается переход человечества к цифровому образу жизни. Революция цифровизации охватила все стороны жизни человечества: промышленность и торговлю, науку и образование, быт и социальные отношения. Выдвинутый 40 лет назад известным российским ученым Андреем Ершовым броский лозунг

«программирование – вторая грамотность» сегодня воплощается в жизнь в форме чуть более точного тезиса: «программирование – одна из обязательных грамотностей 21 века».

2. Как люди учились арифметике последние две с половиной тысячи лет и как будут учиться алгоритмике ближайшие 50 лет

Арифметике и математике уже несколько тысяч лет, десятичной позиционной системе счисления - две с половиною тысячи лет, и столько же лет люди учились преподавать арифметику и математику, изобретали и внедряли счетные палочки и абаки, придумывали занимательные задачи (см. Рис.1).

Рис. 1. Придумки педагогов, преподающих арифметику

Можно утверждать, что методика преподавания арифметики достаточно устоялась и в России, и во всем мире. Знакомство с цифрами, числами и счетом начинается в дошкольном возрасте, систематически арифметика изучается в начальной школе, и каждый выпускник начальной школы на планете Земля сегодня владеет азами арифметики.

Алгоритмика и программирование гораздо моложе арифметики математики. Базовые элементы алгоритмики, подобно базовым элементам арифметики - цифрам и операциям сложения и вычитания - могут быть освоены в раннем детстве, если только детям будет предоставлена материальная среда, позволяющая освоить эти элементы в деятельностно-игровой форме. Человечество сегодня находится в процессе создания подобной среды. Эта материальная среда оказывается более сложной, чем среда для освоения элементов счета. Эта сложность вызвана объективными причинами – описание процессов составления программы человеком и последующего выполнения программы автоматическим устройством требует введения не менее дюжины базовых понятий [1-2], в то время как для описаний процессов счета и использования его результатов можно обойтись гораздо меньшим количеством понятий.

В 21 веке ученые и педагоги всех стран мира пришли к единому мнению о том, что оптимальный путь освоения азов программирования ребенком – составление программ управления реальными и виртуальными роботами. Постепенно вырабатывается консенсус и по вопросу о том, что начинать знакомство с программированием нужно в дошкольном возрасте. Однако вопрос о том, каких роботов и по каким методикам нужно использовать, остается открытым, равно как и вопросы о том,

- каковы цели ознакомления детей с программированием, и насколько систематичным должно быть это знакомство,
- при каких условиях нужно комбинировать в начальном курсе программирования для дошкольников работу в реальных и виртуальных средах, и при каких условиях можно обойтись только виртуальной средой.

3. Мировой опыт

В поисках ответов на эти вопросы в мире создано несколько популярных учебных бестекстовых сред программирования и робототехнических наборов, рассчитанных на детей возраста 5-8 лет.

3.1. Scratch Junior

Сегодня это самая популярная среда бестекстового программирования (см. Рис.2).

Рис. 2. Экран бестекстовой среды программирования **Scratch Junior**.

В нижней строке показаны пиктограммы команд персонажа Котенок

Персонажи, события и процессы, моделируемые в среде Scratch Junior и показываемые на экране, носят игрушечный характер, развиваются в воображаемых, условных двумерных обстановках и потому задача материального воплощения этих обстановок разработчиками Scratch Junior не ставилась. Дети, начинающие знакомиться с программированием в среде Scratch Junior, с первых шагов погружаются в воображаемый виртуальный мир и, как правило, остаются в нем в течение всего периода обучения. Как ис-

ключение, в среде Scrath Junior можно составлять программы, управляющие несложными роботами-игрушками (см. ниже проект Роббо), но без взаимодействия этого робота с виртуальными обстановками и персонажами Scratch Junior. Так что составление программ управления реальными роботами-игрушками в Scratch Junior радикально отличается от программирования воображаемых двумерных персонажей, ради чего и создавался Scratch Junior

3.2. Lightbot (Фонарщик)

Замечательная головоломка, способная увлечь ребенка и мотивировать к изучению программирования (см. Рис.3)

.

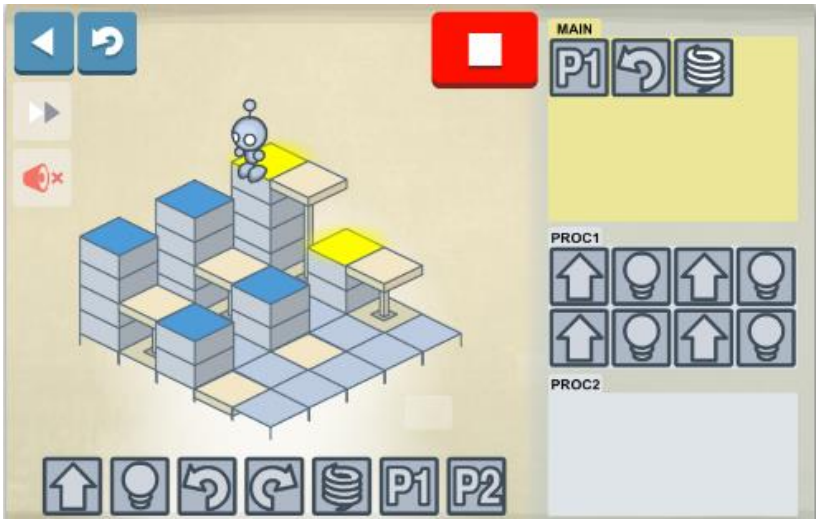


Рис. 3. Экран бестекстовой среды программирования Lightbot. Требуется составить программу, по которой робот, прыгая по блокам, зажжет все блоки, помеченные синим цветом. На данном уровне игры задан шаблон программы, состоящий из главной части программы (main) и двух подпрограмм (proc1 и proc2)

Игра Lightbot позволяет освоить такие концепции, как система команд робота, и подпрограмма, однако не охватывает все доступные дошкольникам базовые концепции программирования. Виртуальная среда, в которой «живет» Фонарщик, сложна, и попытки ее реализации в реальном мире никем не проводились.

3.3. Робомышь

Замечательный робототехнический набор, позволяющий детям освоить концепции программного управления (см. Рис.4).

На корпусе реального робота-игрушки Робомышь размещены которой 7 кнопок с командами, которые умеет выполнять робот. Ребенку показывается игровое поле, на котором размещены Робомышь, препятствия и «кусочкек сыра». Ребенок должен придумать последовательность команд, которые нужно дать Робомыши, чтобы она дошла до сыра. Придуманную последовательность команд – программу движения Робомыши – ребенок с помощью карточек с пиктограммами команд выкладывает на столе и затем «загружает» эту программу в память Робомыши, нажимая на кнопки на ее корпусе. Далее Робомышь самостоятельно проходит запланированный ребенком маршрут.

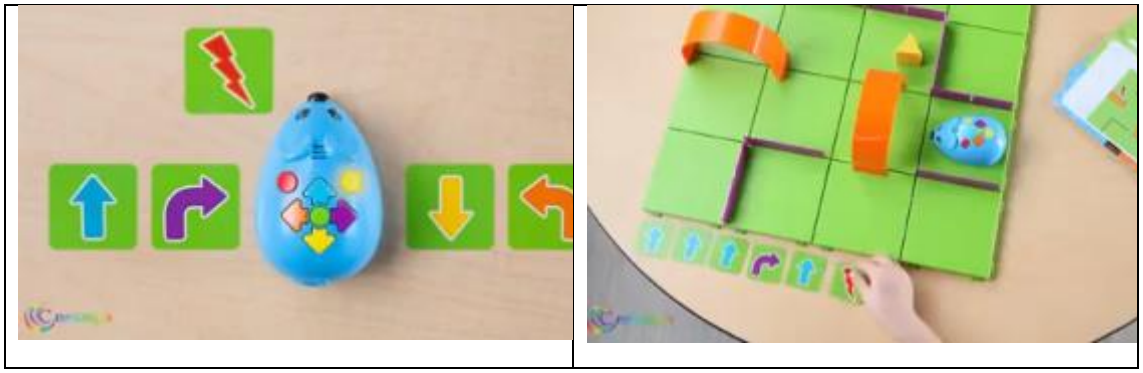


Рис. 4. Слева робомышь, на корпусе которой 7 кнопок с командами, справа – игровая обстановка и программа из карточек, которую выкладывает ребенок

Данный робототехнический набор помогает детям освоить общую идею программного управления, но только на простейших примерах. Еще одним недостатком данного набора, затрудняющим освоение детьми идеи программного управления, является тот факт, что «загруженная» в память мыши программа не наблюдаема детьми. Наконец, с помощью Робомыши нельзя познакомиться с концепциями подпрограмма, ветвление, цикл. То есть Робомышь полезна и эффективна, но только на первых шагах знакомства с программированием.

3.4. Matatalab

Прекрасный робототехнический набор, позволяющий создавать и выполнять программы в материальном мире (см. Рис.5).

Рис. 5. Слева – управляющая башня набора и выложенная на ее пьедестале программа.
Справа – игровое поле, по которому движется управляемый башней робот («водитель» виден через прозрачный оранжевый купол водительского отсека). Выше игрового поля показана крупным планом фрагмент управляющей программы – подпрограмма с именем f0.

Набор Matatalab позволяет составлять программы управления без всякого компьютера, перемещая в материальном мире карточки, на которых изображены пиктограммы команд. По нажатии на кнопку пуск, компьютер управляющей башни «смотрит» на программу, «загружает» ее в свою память и выполняет. Набор позволяет освоить концепции системы команд робота, программного управления и подпрограммы, набрать опыт составления программ. К недостаткам набора следует отнести бедный набор поддерживаемых базовых конструкций программирования, что не позволяет с помощью данного набора освоить весь набор конструкций структурного программирования.

3.5. Робототехнический набор Lego We Do 2.0

Данный набор хорош своей интегрированностью, он включает и компоненты для сборки простейших роботов, и смешанную пиктограммно-текстовую среду программирования собранных роботов. Однако эта среда программирования специализирована, содержит много сложных команд, позволяющих управлять аппаратными компонентами, включенными в набор. Тем самым набор нацелен на изучение детьми

функций и взаимосвязей основных компонент робототехнических систем, а не на освоение основ программирования на примере робототехнических систем. Поэтому набор хорош для изучения азов робототехники детьми возраста 8-12 лет, но для систематического освоения программирования неэффективен.

3.6. Роббо

Робототехнический набор, состоящий из мобильной расширяемой колесной платформы и так называемой Роббо-лаборатории (см. Рис.6).

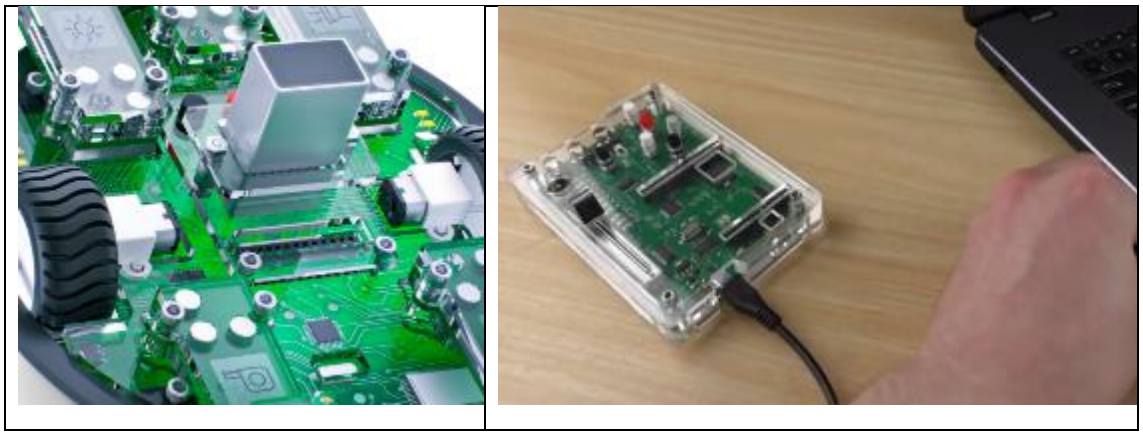


Рис. 6. Слева – колесная платформа Роббо, справа – Роббо-лаборатория

При использовании платформы Роббо дети фактически имеют дело с одним единственным, готовым, уже собранным колесным вездеходом. Никакая деятельность по механической сборке движущегося робота, подобная сборке различных роботов Lego из типовых механических элементов, в наборе Роббо не предусмотрена. К готовой колесной платформе ребенок может подключить с помощью быстросъемных магнитных разъемов один, два или более датчиков, включенных в набор его изготовителем. Движением платформы с использованием информации от установленных на ней сменяемых датчиков можно управлять с помощью программы, созданной в учебной среде программирования Scratch или даже на производственном языке программирования C++. Роббо-лабораторию можно подключить по USB-кабелю к любому компьютеру и превратить в пульт управления мобильной платформой или любыми персонажами программной среды Scratch.

3.7. ПиктоМир

Бестекстовая учебная среда программирования, разработанная по заказу Академии Наук РФ для организации систематического знакомства дошкольников с алгоритмикой (азами программирования), будет подробно описана в следующем разделе (см. Рис.7).

Рис. 7. Решение в среде ПиктоМир задачи по программированию демоверсии ОГЭ по информатике для 9 класса

4. Отечественная сетевая инновационная площадка ПиктоМир по разработке методики, программных средств и учебных пособий для преподавания азов программирования дошкольникам и младшим школьникам

Данная сетевая площадка развернута в 2020 году Академией Наук России совместно с Самарским Институтом Образовательных Технологий и по состоянию на май 2023 года охватывает около 700 детских садов и начальных школ России из 54 субъектов РФ (см. Рис.8).

Отечественный подход к преподаванию алгоритмики (азов программирования) дошкольникам во многом определяется тем, что дошкольное и школьное образование в России централизовано и на 70% проводится по федеральным стандартам и программам и потому может и должна быть поставлена задача интеграции дошкольного курса алгоритмики с школьными стандартами и программами. Из этого следует, что знакомство с азами программирования на дошкольном уровне образования целесообразно сделать достаточно глубоким и систематическим – усилия по освоению программирования на дошкольном уровне образования облегчат всем школьникам освоение программирования на последующих ступенях обучения, а способным школьникам позволят раньше выбрать естественно-техническую специализацию в системе дополнительного образования.

Тем самым, перед дошкольным уровнем образования может быть поставлена задача освоения азов программирования в объеме, охватывающем полный набор конструкций структурного программирования и их наиболее часто встречающихся на практике сочетаний. Число таких сочетаний около 20, и если принять, что каждое сочетание должно быть подкреплено десятком примеров, то получится, что в дошкольном курсе программирования каждый ребенок должен самостоятельно составить около 200 простейших программ.

Двенадцатилетний опыт работы ФГУ ФНЦ НИИСИ РАН в детских садах городов Москва и Сургут и опыт работы с 2020 года с десятками тысяч детей и несколькими сотнями детских садов в рамках сетевой инновационной площадки ПиктоМир показывает, что на освоение азов программирования с такой глубиной требуется порядка 60 занятий при условии проведения одного занятия в неделю, то есть требуется организация двухлетнего курса.

За три года работы на площадке ПиктоМир накоплен опыт организации и проведения систематических пропедевтических курсов алгоритмики и программирования для дошкольников и младшеклассников. Наш опыт показывает, что за

два года занятий по одному разу в неделю в неделю, все без исключения дошкольники с охотой и без затруднений набирают опыт самостоятельного составления и отладки около двух сотен программ и осваивают в отечественной бестекстовой среде программирования ПиктоМир все базовые конструкции структурного программирования и методику использования счета в программах управления реальными и виртуальными роботами.

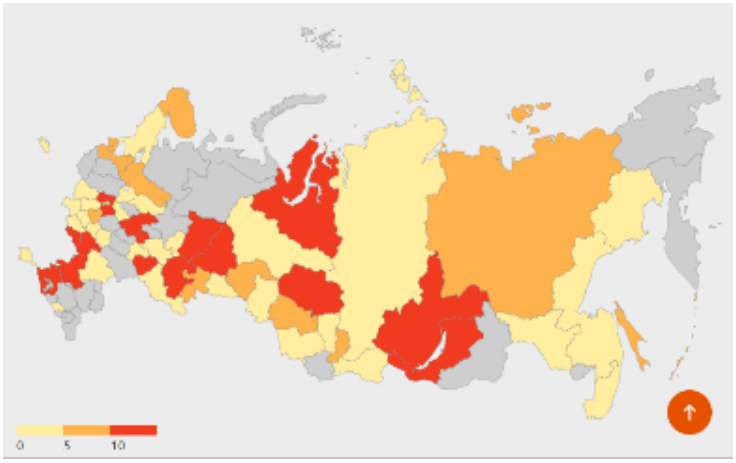
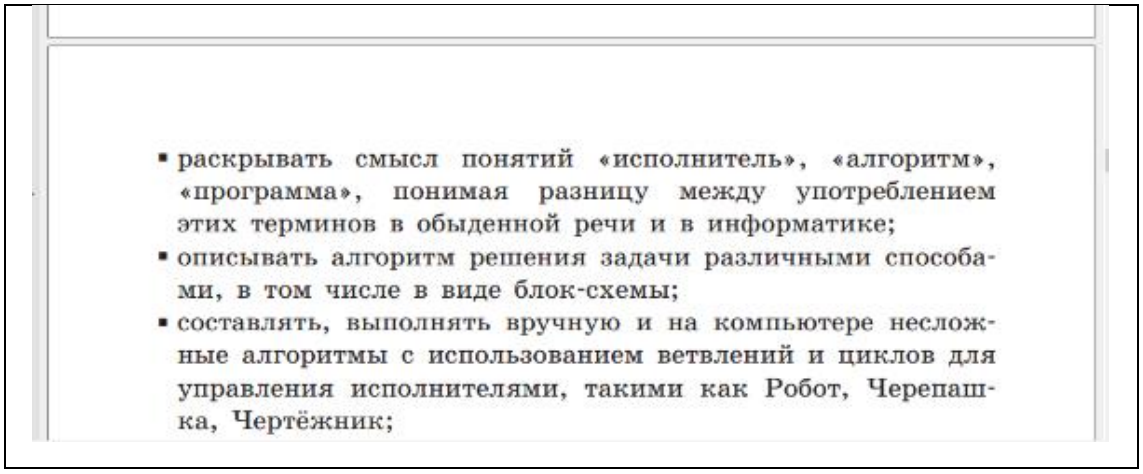


Рис. 8. Распределение по регионам участников инновационной площадки ПиктоМир

Наш опыт показывает также, что прохождение двухгодового дошкольного курса алгоритмики (год в старшей группе и год в подготовительной группе) обеспечивает достижение учащимися ряда существенных результатов (см. Рис. 9) освоения темы программирование, сформулированных в документе «РАБОЧАЯ ПРОГРАММА ОСНОВНОГО ОБЩЕГО ОБРАЗОВАНИЯ, ИНФОРМАТИКА, БАЗОВЫЙ УРОВЕНЬ» [3]

.



- раскрывать смысл понятий «исполнитель», «алгоритм», «программа», понимая разницу между употреблением этих терминов в обыденной речи и в информатике;
- описывать алгоритм решения задачи различными способами, в том числе в виде блок-схемы;
- составлять, выполнять вручную и на компьютере несложные алгоритмы с использованием ветвлений и циклов для управления исполнителями, такими как Робот, Черепашка, Чертёжник;

Рис. 9. Извлечение из раздела «Результаты 9 класс» документа [3]

5. Методика преподавания алгоритмики дошкольникам с помощью среды ПиктоМир – акцент на действия в реальном мире

Двухгодовой курс алгоритмики для дошкольников начинается в старшей группе, для детей возраста около 5 лет. Для этого возраста важна наглядность и вещественность учебных пособий. Поэтому в отличие от подхода, развиваемого разработчиками Scratch Junior, российский подход делает акцент на объекты и действия роботов и детей в материальном, а не в экранно-виртуальном мире. Среда ПиктоМир поддерживает пультовое и программное беспроводное управление реальными роботами-игрушками, перемещающимися по реальным игровым полям, составленным из сочленяемых ковриков. а виртуальные роботы, подобно тому, как это происходит в современной науке и промышленности, вводятся как цифровые двойники реальных роботов с целью облегчения процесса их разработки и эксплуатации.

Более того, сами процессы программирования и выполнения программы на начальной стадии знакомства дошкольников с программированием, выносятся в реальный мир (подобно тому как это сделано в среде Мататалаб, только более удобно методически и технологически). Программы составляются из материальных объектов (англ. tangible objects): деревянных кубиков или магнитных карточек с нанесенными на них пиктограммами команд. Место выкладывания программы не привязано к управляющей башне, а может быть выбрано произвольно. Дети, начиная с возраста 4 года, без затруднений механически размещают кубики или карточки на плоской поверхности рабочего стола или магнитной доски, составляя из них программу по определенным правилам. Такую программу ребенок может выполнить самостоятельно, командуя реальным роботом звуковыми командами с помощью пульта, либо может показать эту программу компьютеру (планшету) воспитателя, после чего планшет выполнит эту программу, командуя роботом на глазах у ребенка и подавая роботу звуковые команды, слышимые и понимаемые ребенком (см. Рис.10).

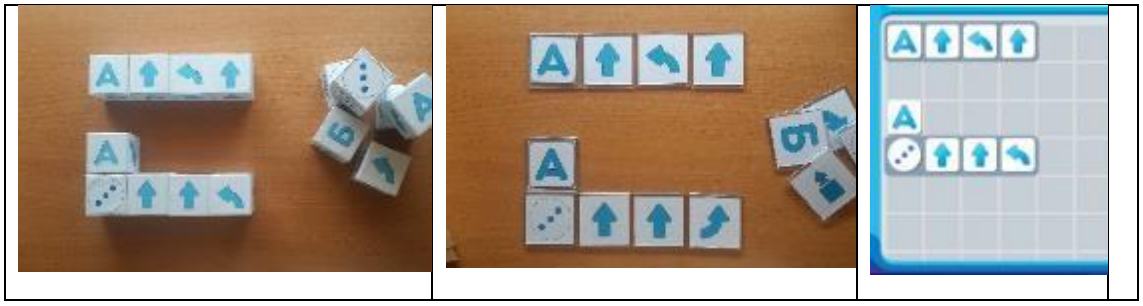


Рис. 10. Программа, выложенная ребенком из кубиков (карточек) на столе и «показанная» ребенком планшету воспитателя и та же программа, «понятая» планшетом, на экране планшета

По методике, разработанной в ФГУ ФНЦ НИИСИ РАН, обучение дошкольников алгоритмике проводится с использование учебного набора, содержащего (см. Рис.13)

- средства создания детьми реальных игровых полей для роботов игрушек (собираются из сочленямых разноцветных ковриков),
- средства создания детьми программ из деревянных кубиков и магнитных карточек,
- мягкие игрушки, имитирующие реальных и виртуальных роботов,
- дистанционно управляемые реальные роботы-игрушки (см. Рис.11, 12).

Рис. 11. Реальный робот Ползун и его звуковой пульт управления; Ползун на реальном игровом поле движется под управлением программы, составленной из кубиков и загруженной в память планшета системой ПиктоМир

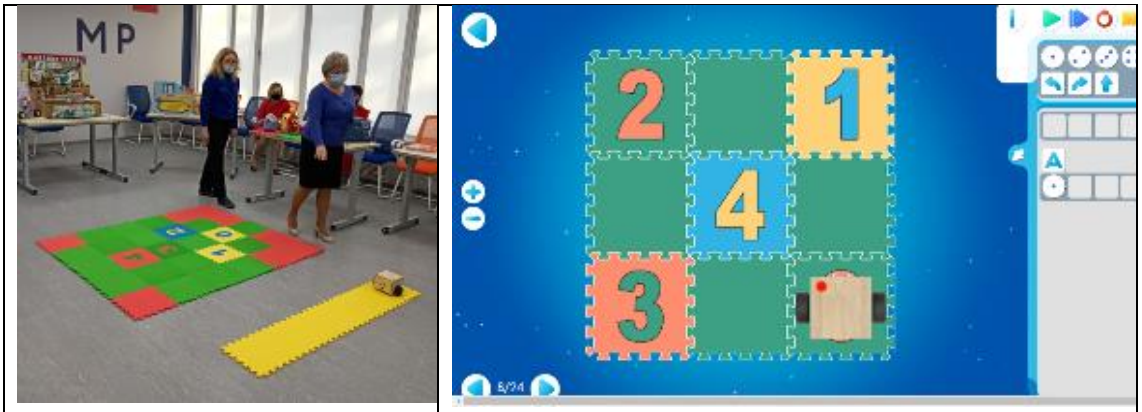


Рис. 12. Ползун на реальном игровом поле из ковриков и его цифровой двойник на экране среды программирования ПиктоМир

К 2020 году, началу совместной деятельности РАН и самарского ИОТ, по результатам десятилетнего преподавания алгоритмики десяткам тысяч дошкольников, Академия Наук России отобрала и опробовала систему основных понятий программирования, доступных для освоения дошкольниками, и разработала методику и материальную среду, позволяющую освоить эти понятия в деятельностно-игровой форме.

.

Рис. 13. Учебный набор «ПиктоМир»

6. Необходимость добавления в учебный набор ПиктоМир роботов, умеющих отвечать на вопросы

И набор понятий, изложенный в статьях [2-3], и поддерживающая его материальная среда (Учебный набор «ПиктоМир», Рис. 13) ориентированы в первую очередь на так называемое программирование без обратной связи. Эта ограниченность легко объяснима. Дело в том, что эти статьи суммируют опыт преподавания алгоритмики в подготовительных группах детских садов в годичном курсе «Алгоритмика для дошколят». Хотя созданная в РАН учебная бестекстовая среда программирования «ПиктоМир» в полной мере поддерживает весь базовый набор конструкций структурного программирования, в том числе и инструменты программирования с использованием обратной связи, в годовой курс для дошкольников (при условии проведения в курсе «Алгоритмика для дошколят» одного 30-минутного занятия в неделю) программирование с обратной связью просто-напросто не умещается. За год занятий дети успевают самостоятельно составить и отладить 100+ простейших программ управления роботами и освоить все конструкций структурного программирования, которые нужны для составления программ без обратной связи. Конструкции ветвление и цикл пока демонстрируются детям на последней паре занятий, однако поработать с ними детям не удается. Можно сказать, что годовой курс алгоритмики для дошкольников обрывается на самом интересном месте.

При работе инновационной площадки выяснилось, что ввиду того, что учебный набор не содержит реальных роботов, позволяющих наглядно продемонстрировать детям обратную связь между компьютером и роботом путем выполнения команд-вопросов, освоение конструкций ветвление и цикл ПОКА требует бóльших усилий, чем мы предполагали при разработке двухлетнего курса. Это недоработка, которую следует устранить, заказав изготовителям учебного набора нового робота-игрушку, умеющего отвечать на вопросы (умеющих выполнять не только команды-приказы, но и команды вопросы) и добавив в цифровую среду ПиктоМир цифрового двойника этого нового робота.

7. Добавление в среду ПиктоМир нового работа, умеющего отвечать на вопросы

7.1. Команды-приказы, команды-вопросы и термин обратная связь в программировании

И в наших курсах, и в других курсах программирования для новичков обучение, как правило, ведется путем составления программ управления роботами с простыми, компактными системами команд. Например, реальный робот Ползун из учебного набора и его цифровой двойник в среде ПиктоМир имеет всего 3 команды: «вперед», «налево» и «направо», эти команды называются командами-приказами. Команды подаются реальному Ползуну звуковыми сигналами. Реальный робот Ползун функционирует в игровой среде, составленной из сочленяемых разноцветных ковриков, причем на некоторых ковриках имеются цифры от 0 до 9. Цифровые двойники этих реальных полей и перемещающийся по ним цифровой двойник реального Ползуна изображаются на экране ПиктоМира. Сигналы реальному Ползуну может подавать человек-Командир с помощью звукового пульта, или компьютер, в память которого помещена (программисты говорят «загружена») программа управления роботом. Получив команду-приказ, реальный Ползун делает следующее:

- повторяет название команды голосом,
- выполняет команду,
- говорит «ГОТОВО»,

после чего готов к выполнению следующей команды. Между компьютером и роботом устанавливается связь в двух направлениях. Направление от компьютера к роботу условно называется «прямым», а направление от робота к компьютеру – «обратным». При выполнении команды-приказа информация пересылается в прямом направлении.

Ползун и его цифровой двойник – очень простые роботы. Программа для Ползуна логически представляет собой длинную последовательность команд «вперед», «налево» и «направо». В ПиктоМире мы можем записать эту длинную последовательность не в одной строке, а в нескольких строках или можем «зашифровать» эту последовательность, сделать ее короче с помощью конструкций повторитель или вспомогательный алгоритм. Но суть программы от этого не изменится, по такой программе компьютер всегда выдаст Ползуну одну и ту же последовательность команд.

Мы хотим включить в учебный набор нового реального робота, который умеет отвечать на вопросы. По получении команды-вопроса новый робот не будет производить никаких движений, а будет только исследовать обстановку вокруг себя, и сообщать компьютеру информацию об этой обстановке, то есть будет отвечать на вопрос компьютера. В этом диалоге информация будет путешествовать сначала в прямом направлении, от компьютера к роботу, а потом в обратном. Поэтому программирование роботов, умеющих отвечать на вопросы, называют программированием с использованием обратной связи.

7.2. Какой новый робот нам нужен

Нового робота мы назовем Толкун – он будет толкать грузы по игровому полю. Это игровое поле будем собирать из тех же ковриков учебного набора, что используются для игровых полей Ползуна. Только на игровом поле для Тягуна дополнительно будут расставлены грузы – ящики с квадратным основанием - на каждый из которых тем или иным способом будет нанесена цифра от 0 до 9. Десяток таких грузов, предположительно изготовленные из картона или пластика, также нужно будет включить в учебный набор.

Детям будет предложено составить программу управления роботом Толкуном, при выполнении которой Толкун переместит каждый груз в клетку с той же цифрой, что и написанная на грузе. У Толкуна будут 5 команд-приказов:

«вперед», «налево», «направо», «мигнуть» и «толкать».

и 2 команды-вопроса

«впереди свободно?»,
«впереди груз?».

Предположительно, производитель сможет изготовить робота Толкуна доработкой уже существующего робота Ползуна, добавив к Ползуну дальномер - ультразвуковой датчик расстояния до препятствия.

По команде «толкать» Толкун будет первым делом проверять ультразвуковым дальномером, есть перед ним груз, и если груза нет – сообщит, что команда невыполнима. Если же впереди Толкуна груз есть, то он подойдет к нему вплотную, запомнив, какое расстояние он преодолел, затем будет толкать груз в следующую клетку поля на расстояние, равное размеру клетки игрового поля, а потом задним ходом вернется в центр той клетки, в которой ранее стоял груз. По завершении выполнения команды Толкун скажет «готово» и будет готов к получению следующей команды.

Получив команду-вопрос «впереди груз?», Толкун ультразвуковым дальномером измерит расстояние до ближайшего препятствия. Если это расстояние окажется меньше ширины клетки игрового поля, то Толкун ответит компьютеру ДА, а в противном случае ответит компьютеру НЕТ. На команду-вопрос «впереди свободно?» Толкун ответит НЕТ, если впереди груза нет, и ответит ДА в противном случае. Наконец, получив команду «вперед», Толкун будет первым делом проверять ультразвуковым дальномером, есть ли перед ним груз, и если груза есть – сообщит, что команда невыполнима, а если груза нет – продвинется вперед на одну клетку.

7.3. Как и зачем дети будут играть с роботом Толкуном

Игры с роботом Толкун должны подготовить детей к программированию с обратной связью, к которому дети подготовительной группы приступают во второй половине второго года обучения. Толкун умеет «разговаривать»: при включении сообщает свое Имя и выражает готовность к работе, а если про него «забыть» на несколько минут, «обидится» и скажет, что выключается.

Команды роботу Толкуну подаются, как и роботу Ползуну с помощью звукового кодирования, каждая команда кодируется проигрыванием определенных сигналов.

Команды-приказы:

- команда *«вперед»* —·● — 1 бип (короткий звук, **точка** азбуки Морзе)
- команда *«налево»* — ● ● — 2 бипа (2 коротких звука)
- команда *«направо»* — ● ● ● — 3 бипа (3 коротких звука)
- команда *«толкать»* — ● ● ● ● — 4 бипа (4 коротких звука)
- команда *«мигнуть»* — ● ● ● ● ● — 5 бипов (5 коротких звуков)

Команды-вопросы:

- команда *«впереди свободно?»* — ▬ — 1 бип (1 длинный звук. **тире** азбуки Морзе)
- команда *«впереди груз?»* — ▬ ▬ — 2 бипа (2 длинных звука)

Команды Роботу могут отдаваться либо Командиром с помощью звукового пульта управления (смартфоном, на котором установлено новое приложение ПМ Пульт для Толкуна) либо компьютером (на котором установлена новая версия ПиктоМира, поддерживающая работу с роботом Толкуном).

Кто бы ни командовал роботом Толкуном, дети могут слышать, как команда подается, и видеть (и слышать), как робот ее выполняет. При выполнении каждой команды, радиоуправляемый робот Толкун, подобно роботу Ползуну, должен сообщать детям голосом, какую команду он выполняет.

Замечание. Для надежности, слышимые детьми звуковые сигналы и ответы Толкуна ДА или НЕТ должны дублироваться радиосигналами, как это уже сделано в Ползуне.

Пульт со звуковыми командами можно использовать в играх, в которых дети распределяют роли: Робот, Командир, Программист. Робот выполняет команды, двигаясь по коврикам игрового поля; Командир нажимает на кнопки на экране пульта; Программист выкладывает программу из кубиков.

Толкун материален, его можно потрогать, пощупать, покрутить в руках. Он может выполнять команды и по каждой команде выполнять понятное действие. Команды Толкуну даются не словесно, а звуковыми сигналами. Подобно солдату, получив команду, Толкун говорит, что он будет делать по этой команде, а по окончании выполнения команды докладывает «готово».

В случае, если Толкун не может выполнить полученную команду-приказ, он сообщает об этом командиру. Если, например, Толкун получил команду *«вперед»*, а в клетке перед Толкуном стоит груз, Толкун сообщает: «Команда невыполнима. Впереди груз». А если Толкун получил команду *«толкать»* при отсутствии груза, он сообщает «Команда невыполнима. Нéчего толкать».

Пульт Толкуна, похож на пульт Ползуна, но дополнен командами-вопросами (см. Рис.14). При выполнении команд-вопросов Толкун выдает голосом такие сообщения:

«отвечаю на вопрос *«впереди свободно?»*. Ответ: ДА»

«отвечаю на вопрос *«впереди свободно?»*. Ответ: НЕТ»,

«отвечаю на вопрос *«впереди груз?»*. Ответ: ДА»,

«отвечаю на вопрос *«впереди груз?»*. Ответ: НЕТ»

Толкун реагирует только на те звуковые команды, которым его «научили» на заводе-изготовителе. Чтобы ребенок смог изображать нового робота, с которым его знакомят, нужно показать ребенку, как Толкун выполняет звуковые команды-приказы и команды-вопросы, и какие слова он при этом говорит. При этом нужно не забыть показать и такие ситуации, в которых Толкун не может выполнить полученную команду-приказ. Научив ребенка изображать робота, можно дать ему повязку «робот», а другому ребенку – повязку «командир». После этого «командир» может начать командовать «роботом».

Еще интереснее, если рядом с «роботом-ребенком» поставить настоящего робота. По командам «командира» оба робота будут выполнять одни и те же движения, говорить одни и те же слова. Если ребенок научился командовать роботом так, чтобы робот прошел по какому-то маршруту, то команды, которые проводят робота по этому маршруту можно запомнить, составив программу.

Программа позволяет запомнить какие команды и в каком порядке нужно давать роботу, чтобы он прошел по данному маршруту. Программу можно составлять из «пикто-квадратиков» или из «пикто-кубиков». На гранях квадратика или кубика изображены движения, которые должен делать робот: *«вперед»*, *«налево»*, *«направо»*, *«толкать»*, *«мигнуть»*. С правилами составления простейших программ дети познакомились, играя с Ползуном. Но при составлении программ для Толкуна нужно освоить пиктограммы команд-вопросов и их использование в конструкциях цикл ПОКА  и ЕСЛИ 

			
впереди свободно?	*впереди груз*?	ЕСЛИ впереди груз выполнить алгоритм А	ПОКА впереди свободно выполнить команду НАЛЕВО

Рис. 14. Команды-вопросы Толкуна и примеры их использования

7.4. Примеры заданий, выполняемых с роботом Толкуном

Задание на использование конструкции ЕСЛИ. Рядом с Толкуном несколько грузов, сколько именно – неизвестно. Толкун должен каждый из этих грузов отодвинуть на одну клетку и вернуться в исходную позицию (см. Рис.15).

Рис. 15. Решение задания на использование конструкции ЕСЛИ

Идея решения. Сначала составляем алгоритм А, который отодвигает на одну клетку груз, лицом к которому стоит Толкун и возвращает Толкуна в исходную позицию. В главном алгоритме в цикле 4 раза поворачиваем Толкуна и в каждом из его 4-х положений проверяем, есть ли *«впереди груз»*, если ДА, то выполняем алгоритм А, то есть отодвигаем обнаруженный ящик.

Задание на использование цикла ПОКА. Рядом с Толкуном стоит груз, в какую сторону «смотрит» Толкун – неизвестно. Толкун должен сдвинуть груз на одну клетку (см. Рис.16).

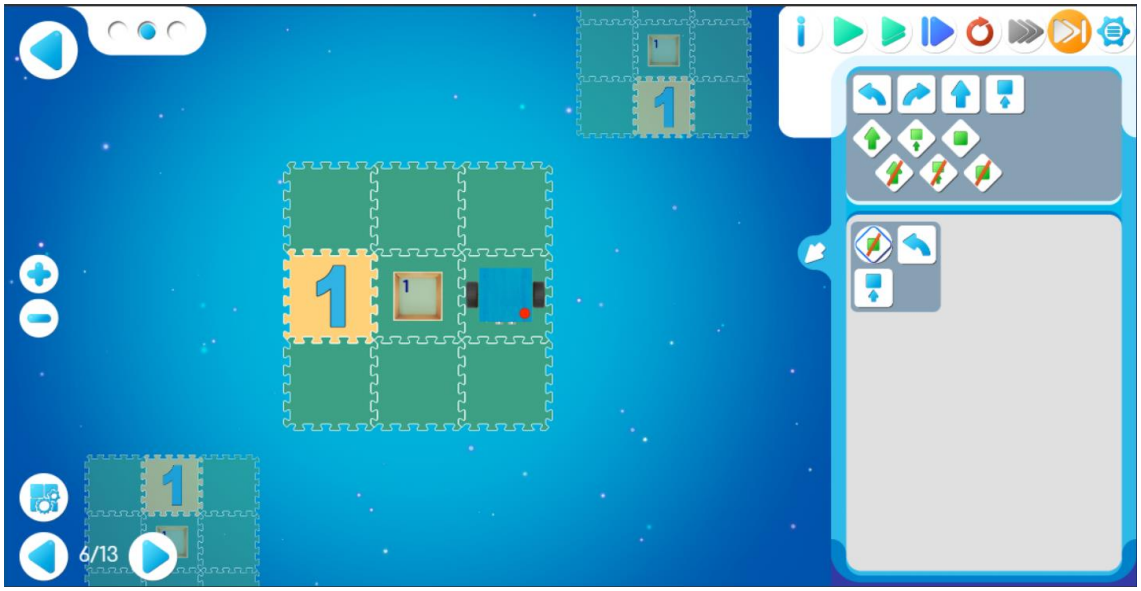


Рис. 16. Решение задания на использование конструкции цикл ПОКА

Идея решения. В программе используется цикл ПОКА. Компьютер задает Толкуну вопрос *«впереди свободно?»*. Получив ответ ДА, компьютер дает Толкуну команду-приказ *«налево»* и снова и снова задает вопрос *«впереди свободно?»*, пока не получит ответ НЕТ. В этот момент выполнение цикла ПОКА завершается и компьютер выполняет следующую команду программы – дает Толкуну команду-приказ *«толкать»*.

8. Повышение вовлеченности детей в процесс выполнения программы роботом Толкуном

Чтобы повысить вовлеченность детей в процесс выполнения программы Толкуном, нужно включить в состав системы команд Толкуна такие команды, которые он будет выполнять с помощью ребенка. Назначив ребенка помощником Толкуна, мы сделаем ребенка полноправным участником и процессов выполнения роботом отдельных команд, и процесса выполнения программы компьютером.

Какие новые команды можно было бы ввести. На игровом поле Толкуна уже расставлены грузы, и задача Толкуна – перемещать эти грузы в нужные места. Можно было бы сделать нового робота Толкун-2, который имеет манипулятор и умеет, подойдя к грузу, подхватить его и поставить себе на крышу. Переместившись по полю в нужное место, Толкун-2 мог бы с помощью манипулятора выгрузить груз с крыши в свободную клетку перед собой. Это позволило бы Толкуну-2 быстрее выполнять многие работы.

Толкуну-2 нужно добавить бы две новые команды-приказа: «погрузить» и «разгрузить». Конструирование и изготовление робота Толкун-2, возможно, но робот получится сложным и дорогим. Чтобы этого избежать, поступим так.

Не меняя механизмов робота Толкуна, научим его понимать еще две команды-приказа:

- команда «погрузить» — ● ▬ ● (точка тире точка)
- команда «разгрузить» — ● ▬ ▬ ● (точка тире тире точка)

По команде *«погрузить»* Толкун-2 должен говорить: *«получил команду погрузить, прошу помощи»* и далее начинает обратный отсчет: *«девять, восемь, семь, шесть, пять, четыре, три, два, один»*. По завершении обратного отсчета Толкун-2 говорит «погрузка завершена, спасибо». Аналогично могла бы выполняться команда *«разгрузить»*. Пока Толкун-2 ведет обратный отсчет, ребенок, назначенный в помощь Толкуну-2, должен успеть провести погрузку или разгрузку Толкуна-2: поставить груз на крышу или снять груз с крыши и поставить его в клетку перед Толкуном. Пользуясь ультразвуковым дальномером, Толкун-2 должен анализировать обстановку перед собой и отказываться выполнять команду *«погрузить»*, когда перед ним свободная клетка и команду *«разгрузить»*, когда перед ним груз.

Представляется, что дети охотно включились бы в описанное выше взаимодействие с Толкуном-2 и быстрее освоили бы идею команд-вопросов и примеры их использования. От изготовителя Толкуна и разработчиков звукового

пульта управления Толкуном добавление двух новых команд потребовало бы только небольшого изменения программного обеспечения.

Описанный выше порядок выполнения Толкуном-2 команд *«погрузить»* и *«разгрузить»* обладает тем недостатком, что время, выделяемое на погрузку или разгрузку фиксировано. Если сделать его слишком большим, это приведет к ненужным паузам в выполнении программы, а если сделать его слишком маленьким, ребенок может не успеть погрузить или разгрузить груз.

От этого недостатка можно было бы избавиться, если бы изготовитель робота придумал способ добавить в конструкцию Толкуна-2 датчик, который определяет, есть ли груз на крыше Толкуна-2 или нет. При наличии такого датчика, Толкун-2 по команде *«погрузить»* говорил бы: *«получил команду погрузить, прошу помочь»* и начинал подавать прерывистый звуковой и световой сигнал. А в тот момент, когда датчик показал, что груз уже установлен, говорил бы ««*погрузка завершена, спасибо»*. Аналогично могла бы выполняться команда *«разгрузить»*.»

Кроме того, наличие подобного датчика позволило бы Толкуну-2 отвечать на команды-вопросы *«багажник занят?»* и *«багажник свободен?»*, а также отказываться выполнять команду *«погрузить»*, если багажник занят и команду *«разгрузить»*, если багажник свободен.

9. Заключение

С появлением компьютеров и цифровых технологий, алгоритмика и программирование стали неотъемлемой частью образования в 21 веке. В статье был проведен обзор и выявлены достоинства и недостатки различных учебных сред, с помощью которых сейчас дети уже в дошкольном возрасте могут осваивать базовые навыки программирования. По опыту обучения дошкольников в среде ПиктоМир, была отмечена необходимость демонстрации детям обратной связи между компьютером и роботом, и предложено решение, заключающееся в создании нового робота, отвечающего на команды-вопросы. Также была проведена работа по добавлению такого робота в среду ПиктоМир и составлены методика и задания с его использованием. Однако необходимо продолжать исследования в области обучения алгоритмики и программирования, чтобы повышать вовлеченность детей в процесс выполнения программы роботом. Для этого можно включить в состав системы команд робота две новые команды, которые он будет выполнять с помощью ребенка.

Работа выполнена в ФГУ ФНЦ НИИСИ РАН в рамках фундаментальных исследований по теме FNEF-2022-0010.

Авторы благодарны К.А. Мащенко и А.В. Шляхову за многочисленные полезные обсуждения.

Initial Algorithmic Training for Preschoolers and Other Beginners with the Help of Smart Robot Toys

A.G. Kushnirenko, A.G. Leonov, D.V. Mashchenko, M.V. Raiko, I.N. Gribanova

Abstract. In the modern world, algorithmics is becoming on a par with school arithmetic, because the basic programming elements mastered by a preschooler will accompany him throughout his life because of global digitalization. The article describes the world practices of teaching algorithmics to children, the pros and cons of various techniques are noted. Special attention is paid to the textless educational programming environment PiktoMir, the experience of organizing and conducting systematic propaedeutic courses of algorithmics and programming for preschoolers and junior high school students is described. The necessity of demonstrating feedback between a computer and a robot to children is described. The authors proposed a solution consisting in adding a new robot to the PiktoMir environment that answers commands-questions. Conclusions are drawn about the direction of further research, in particular, about the need to increase the involvement of children in the process of implementing the program by a robot.

Keywords: algorithmics, PiktoMir, preschool programming course, textless programming environment, smart robots-toys, programming with feedback

Литература

1. РАБОЧАЯ ПРОГРАММА ОСНОВНОГО ОБЩЕГО ОБРАЗОВАНИЯ, ИНФОРМАТИКА, БАЗОВЫЙ УРОВЕНЬ, (для 7–9 классов образовательных организаций) Одобрена решением федерального учебно-методического объединения по общему образованию, протокол 3/21 от 27.09.2021 г.
https://edsoo.ru/Primernaya_rabochaya_programma_osnovnogo_obschego_obrazovaniya_predmeta_Informatika_proekt_.htm
2. В.Б. Бетелин, А.Г. Кушниренко, А.Г. Леонов. Основные понятия программирования в изложении для дошкольников // Информатика и ее приложения, 2020 г. Т. 14. Вып. 3. С. 56-62. DOI: 10.14357/19922264200308
3. Betelin V.B., Kushnirenko A.G., Leonov A.G., Mashchenko K.A. Basic Programming Concepts as Explained for Preschoolers // International Journal of Education and Information Technologies (NAUN). DOI: 10.46300/9109.2021.15.25. Volume 15, 2021, pp. 245-255, E-ISSN: 2074-1316

Особенности использования цифрового следа обучающихся в системах искусственного интеллекта в образовании

М.С. Дьяченко

ФГУ ФНЦ НИИСИ РАН, Москва, Россия, mdyachenko@niisi.ru

Аннотация. В статье рассмотрены особенности сбора и использования данных цифрового следа обучающихся на этапах внедрения новых учебных технологий на основе искусственного интеллекта. Исследуемые особенности характерны для разработки решений, требующих большого объёма предварительно накопленных данных цифрового следа обучающихся. Рассмотрены вопросы ускорения накопления данных, обезличивания данных цифрового следа обучающихся, вопрос обучения модели по федеративной схеме без извлечения данных, рассмотрены схемы исследований без достаточного объема данных, а также подходы к адаптации внедренных решений при переносе их между учебными заведениями или предметными областями.

Ключевые слова: цифровой след обучающихся, интеллектуальные системы обучения, машинное обучение, обезличивание цифрового следа обучающихся, адаптация решений ИИ

1. Введение

Необходимость технологической независимости Российской Федерации становится в наше время стимулом для дальнейшего повышения уровня подготовки выпускников школ и вузов. В системе образования РФ активно готовятся к внедрению в учебный процесс перспективных методов искусственного интеллекта. Одним из признаков такой подготовки можно считать появление стандартов для образовательных продуктов с алгоритмами искусственного интеллекта. Стандарты в том числе охватывают требования по работе с данными, а также требования к контрольным выборкам для испытания систем искусственного интеллекта в образовании. Такое внимание к данным не случайно, поскольку для создания решений на основе современных методов искусственного интеллекта необходимы большие наборы данные об обучаемых, собираемых учебной системой.

Решения на основе ИИ в образовании используются для реализации адаптивного обучения, элементов предсказательной учебной аналитики, систем поддержки принятия решения и автоматической генерации заданий. Проводившиеся в РФ эксперименты по внедрению технологий ИИ в образовании основывались на реализации методов, которые могут быть разработаны без больших объёмов предварительно накопленных данных об обучаемых, например, как в исследовании Кречетова И.А. с использованием генетических алгоритмов и кривых забывания [1], или адаптивных технологий без использования предварительно накопленных данных, как в исследованиях Шамсутдиновой Т.М. [2] и Вайнштейн Ю.В. [3]. Однако с появлением перспективных технологий, например, таких как глубокая трассировка знаний [4], её развития для учёта модели забывания [5] или содержания заданий [6], становится актуальным накапливать и использовать большие объёмы данных об обучении.

Требования к сбору, хранению, обработке, передаче и защите данных для систем ИИ в образовании приводятся в ГОСТ Р 59897-2021. Однако в рассмотренных исследованиях отсутствует достаточно информации об особенностях внедрении в РФ технологий обучения на основе ИИ, использующих для разработки большие объёмы накопленных данных. В связи с этим становится актуальной задача исследования особенностей использования цифрового следа обучающихся в процессе внедрения технологий ИИ в образовании, использующих большие объёмы предварительно накопленных данных.

2. Методология исследования

В исследовании используется описанный в работе автора [7] процесс внедрения новых учебных технологий, разрабатываемых с использованием больших объёмов данных об обучении. Укрупненно процесс состоит из последовательных этапов: накопления данных об обучении, краткосрочной оценки эффективности, долгосрочной оценки эффективности и масштабирования учебной технологии, использующей ИИ. На каждом из этих этапов рассматриваются особенности, вызванные использованием большого объёма данных. Выполнен анализ проведенных различными исследователями результа-

тов внедрения рассматриваемых учебных технологий и использования накопленных данных об обучении.

3. Результаты исследования

3.1. Сбор данных цифрового следа обучающихся

Рассмотрим некоторые особенности сбора данных цифрового следа обучаемых (далее ЦСО), происходящего на этапах подготовки к разработке и оценке эффективности внедряемой учебной технологии, использующей ИИ.

Данные для разработки решений на основе ИИ поступают из внутренних и внешних источников данных. Примеры внешний источников – это социальные сети [8] или библиотечная информационная система. Внешние источники данных должны предоставлять интерфейс для получения данных по результатам использования обучаемым внешней системы, например, это данные поисковых запросов в библиотечной информационной системе или данные диалога обсуждения задания с одногруппниками в социальной сети. При работе с внешними источниками данных необходимо учитывать время гарантированного хранения данных во внешней системе, возможность сопоставления идентификатора обучаемого во внутреннем и внешнем источниках данных, возможность упорядочивания данных внутренних и внешних источников в порядке возникновения событий. Например, если время гарантированного хранения данных во внешней системе меньше необходимого времени накопления данных, то данные из внешней системы следует копировать в хранилище, обеспечивающее необходимое время хранения данных ЦСО. Данные из каждого источника должны обладать метаданными с описанием содержания, технических деталей хранения, процесса обработки и доступа к данным.

В учебных системах, использующих ИИ, регистрации подлежат также результаты работы ИИ решений и действия преподавателя в системе. К этим данным также должны прилагаться метаданные. Для анализа эффективности внедряемого ИИ решения также необходимо собирать результаты работы ИИ алгоритмов (например, прогноз результатов обучения, результаты трассировки знаний – все, что влияет на поведение автоматизированной системы обучения) и регистрировать действия преподавателя, например, такие как ручная корректировка действия, предложенного автоматизированной системой с использованием алгоритма ИИ. Эти данные необходимы для анализа эффективности внедряемого ИИ решения.

Следующей особенностью сбора данных ЦСО для разработки и оценки эффективности является значительная длительность подготовительного этапа, обусловленная ограниченной пропускной способностью экспериментальных курсов. Одной из характеристик данных является их изменчивость – промежуток времени, в рамках которого данные остаются актуальными для целей исследования, например, изменение структуры учебного курса или учебных материалов может привести к потере актуальности данных результатов проверки знаний. С учётом изменчивости данных можно определить необходимую минимальную пропускную способность экспериментальных курсов для сбора требуемого объёма данных в пределах периода их актуальности [7]. Ускорить сбор данных можно за счёт подключения к эксперименту большего количества учебных заведений или запуска параллельного онлайн курса с менторами, который не привязан к расписанию учебного заведения, но при этом повторяет структуру курса учебного заведения.

Следующей особенностью сбора данных ЦСО является необходимость продолжать собирать данные после завершения курса для автоматизации контрольного среза остаточных знаний [9]. Для реализации контрольного среза необходимо собрать данные экспериментального курса и данные одного из следующих за ним курсом, который использует знания, полученные на экспериментальном курсе. В результате становится возможным оценить не только краткосрочный эффект от внедряемой учебной технологии, но и долгосрочный эффект.

К особенностям решений на основе ИИ, использующих большие объёмы накопленных данных, можно отнести необходимость сбора данных после завершения внедрения для обеспечения непрерывной оценки качества решения и доработки решения с целью улучшения его характеристик с использованием вновь полученных данных.

3.2. Использование накопленных данных цифрового следа обучающихся

Рассмотрим некоторые особенности использования накопленных данных ЦСО, которые в первую очередь нужны для разработки (обучения) решения на основе ИИ, а также для анализа данных и оценки эффективности применения разработанного решения.

Данные ЦСО для исследований и разработки можно использовать только в обезличенном виде. Обезличивание данных позволяет убрать из них компоненты, позволяющие однозначно идентифицировать принадлежность блока дан-

ных к конкретному обучаемому. Согласно исследованию Грацского технического университета [10] учебные данные обладают большим количеством перекрестных связей, что затрудняет их обезличивание и создает высокие риски для повторной идентификации. Например, имея доступ к данным учебной группы с временными метками событий можно однозначно определить группу, данные которой зарегистрированы, а анализируя сами данные (например, полученные оценки) сопоставить результаты с данными из других источников, таких как социальные сети и открытые источники информации. В результате появляется возможность однозначно сопоставить обезличенные данные с конкретными обучаемыми. В связи с этим обработка даже обезличенных данных должна выполняться в автоматическом режиме без участия человека. Например, обобщение данных для анализа можно выполнять без извлечения данных, также как и обучение решений на основе ИИ может выполняться по федеративной схеме [11] с использованием данных непосредственно в месте их хранения без необходимости их извлекать или перемещать.

Состав и формат сохраняемых в учебных системах данных ЦСО определяется на этапе разработки, например, это база данных в СДО Moodle [12] или журналы OpenEdu [13], а также возможности по регистрации данных ограничены доступными интерфейсами взаимодействия с внешними системами (библиотечные информационные системы, системы видеоконференций, диалоговые ассистенты преподавателя, социальные сети), которыми обучаемый пользуется в процессе обучения. Хотя учебные системы поддерживают унифицированные форматы регистрации результатов обучения, такие как спецификация Experience API (xAPI, https://xapi.com/), в настоящий момент в системе образования РФ нет единого общепризнанного формата хранения детальных данных ЦСО. Инициатива «Университета 20.35» по созданию стандарта ЦСО (https://standard.2035.university/), вероятно, направлена на решение специфичных задач, поскольку предлагаемый формат ЦСО содержит отсылки к специализированным коммерческим решениям, таким как Trello, Zoom, Discord, Slack. Внедрение общепринятого стандарта хранения данных ЦСО и реализация его в распространённых в РФ системах обучения позволит расширить возможности для исследований за счет охвата большего объёма данных детального ЦСО, а также даст возможность в перспективе создать большие открытые наборы данных для исследований и разработки в области применения ИИ в обучении.

3.3. Проблема недостаточного для разработки объёма данных цифрового следа обучающихся

Процесс накопления больших объёмов данных эффективно решён только в МООК масштаба таких систем, как Coursera (https://www.coursera.org/) и Stepik (https://stepik.org/), или в коммерческих системах обучения, таких как Пларио (https://plario.ru/) и KNewton (https://www.knewton.com/), учебные курсы которых одновременно используются в нескольких учебных заведениях.

Накопление необходимых данных в рамках одного учебного заведения занимает много времени [7], поэтому разработчики вначале запускают систему с автоматизированным учебным курсом, который реализует элементы ИИ, не требующие для разработки использования больших объёмов предварительно накопленных данных, после чего начинается предварительный этап сбора данных, в ходе которого недостающие данные ЦСО накапливаются, опять же за продолжительное время. При этом существуют организационные особенности, ограничивающие возможности по накоплению данных ЦСО в учебном заведении: начальные курсы в учебных заведениях как правило имеют большую пропускную способность поскольку являются частью общей базовой подготовки, через которую проходят все обучаемые, тогда как специализированные курсы при завершении обучения имеют существенно меньший охват.

По причине сложности накопления больших объёмов данных в сжатые сроки исследователи используют следующий подход при внедрении новых технологий [14]: берутся открытые для исследований наборы данных, которые нужны для получения предварительных результатов, после чего выполняют запуск эксперимента в учебных заведениях для накопления реальных данных и оценки эффективности внедряемой технологии. В случае отсутствия доступных для исследования наборов открытых данных формируются синтетические данные с требуемыми характеристиками, которые используются для получения предварительных результатов, после чего также начинается экспериментальная проверка с использованием реальных данных в учебном заведении.

Особенностью этапа масштабирования использования внедряемой технологии является также необходимость повторного сбора данных для разработки (или доработки) и оценки эффективности применения решения с элементами ИИ. Если решение с элементами ИИ зависит от структуры входных данных, например, как решения с использованием трассировки знаний,

зависящее от структуры предметной области, то изменение учебной программы или корректировка учебных материалов приведёт к необходимости доработки решения или к необходимости повторной разработки решения, начиная с этапа накопления данных. Для упрощения распространения решений на основе ИИ разрабатываются методы адаптации решения к новым условиям применения – адаптация предметной области. Например, для решений на основе глубокой трассировки знаний разработан алгоритм для переноса реализации из одного учебного заведения в другое с учётом изменения структуры и содержания данных [14], примечательно, что для переноса требуется меньше данных, чем для повторной разработки решения на данных нового учебного заведения.

4. Заключение

Наличие больших объёмов накопленных данных ЦСО является предусловием для обеспечения возможности исследования и внедрения в системы автоматизации учебного процесса перспективных методов искусственного интеллекта.

Накопление данных ЦСО в рамках одного учебного заведения – длительный процесс из-за ограниченной пропускной способности экспериментальных курсов. Для ускорения накопления данных ЦСО в эксперименте должны участвовать несколько учебных заведений, что возможно при создании гетерогенной экспериментальной среды или при использовании централизованной системы обучения, внедрённой в учебные процессы нескольких учебных заведений. Перспективным направлением является создание открытых больших наборов данных для исследований и разработки новых решений в области применения ИИ в образовании.

Использование накопленных данных ЦСО возможно только в обезличенном виде, при этом обезличивание данных ЦСО учебного заведения является сложной процедурой из-за большого количества перекрестных связей. По этой причине для учебных данных высок риск повторной идентификации (восстановление привязки к личности по обезличенным данным). В связи с этим необходимо исключать возможность выгрузки даже обезличенных данных в процессе исследований и разработки, выполняя процессы анализа данных и разработки в автоматическом режиме без участия человека. Например, за счет использования методов федеративного обучения решений с использованием ИИ, при котором распределенно хранимые данные используются непосредственно в месте их хранения без необходимости извлекать или перемещать данные на время разработки.

Внедренные решения с элементами ИИ не обязательно являются универсальными и могут зависеть от структуры входных данных. В результате этого при переносе разработанного решения в другое учебное заведение или при изменении материалов учебного курса необходимо дорабатывать решение за счет использования актуальных данных ЦСО. Для исключения необходимости повторять весь процесс накопления данных ЦСО и следующей за ним повторной разработки, исследуются методы по смене предметной области, применения которых позволит, собрав небольшой объём актуальных данных, адаптировать созданное решение для использования в новой предметной области.

Работа выполнена в ФГУ ФНЦ НИИСИ РАН в рамках фундаментальных исследований по теме FNEF-2022-0010.

Specificities of Using the Digital Footprint of Students in Artificial Intelligence Systems in Education

M.S. Dyachenko

Abstract. The article discusses the features of collecting and using digital footprint data of students at the stages of introducing new educational technologies based on artificial intelligence. The studied specificities are common for the development of solutions that require a large amount of pre-accumulated data of the digital footprint of students. The issues of accelerating data accumulation, depersonalization of the digital footprint of students, the issue of using a federated learning without data extraction, research and development schemes without a sufficient amount of data, as well as approaches to adapting implemented solutions when transferring them between educational institutions or subject areas are considered.

Keywords: digital footprint of students, intelligent learning systems, machine learning, depersonalization of the digital footprint of students, AI solutions adaptation

Литература

1. И. А. Кречетов, В. В. Романенко Реализация методов адаптивного обучения // Вопросы образования. 2020. № 2. С. 252–277.

2. Т. М. Шамсутдинова. Формирование индивидуальной образовательной траектории в адаптивных системах управления обучением // Открытое образование. 2021. Т. 25. № 6. С. 36–44.

3. В. А. Шершнева, Ю. В. Вайнштейн, Т. О. Кочеткова. Адаптивная система обучения в электронной среде // Программные системы: теория и приложения. 2018. Т. 9. № 4 (39). С. 159–177.

4. C. Piech и др. Deep Knowledge Tracing., NIPS, 2015.

5. W. Zhao и др. A novel framework for deep knowledge tracing via gating-controlled forgetting and learning mechanisms // Information Processing & Management. 2023. Т. 60. № 1. С. 103-114.

6. Q. Liu и др. EKT: Exercise-Aware Knowledge Tracing for Student Performance Prediction // IEEE Transactions on Knowledge and Data Engineering. 2021. Т. 33. № 1. С. 100–115.

7. М. С. Дьяченко, А. Г. Леонов. Архитектура учебной системы с индивидуализацией обучения на основе накопленных данных результатов автоматизированной проверки заданий // Успехи кибернетики. 2023. Т. 4. № 1(13). С. 39–48.

8. E. Pozdeeva и др. Assessment of Online Environment and Digital Footprint Functions in Higher Education Analytics // Education Sciences. 2021. Т. 11. № 6. С. 256.

9. С. М. Григорьев и др. Анализ сущности и содержания контроля успеваемости обучающихся // Национальная ассоциация ученых. 2015. № 4– 3 (9). С. 102–104.

10. M. Khalil, M. Ebner De-Identification in Learning Analytics // Learning Analytics. 2016. Т. 3. № 1.

11. C. Fachola и др. Federated Learning for Data Analytics in Education // Data. 2023. Т. 8. № 2. С. 43.

12. Т. А. Кустицкая и др. Цифровой след из LMS Moodle для прогнозирования результатов обучения // Информатизация образования и методика электронного обучения. 2022.

13. Н.Д. Барсуков и др. Анализ активности студентов на курсах онлайн-обучения на основе логов платформы «OpenEdu» // Труды Института системного программирования РАН. 2020. Т. 32. № 3. С. 91–100.

14. S. Cheng и др. AdaptKT: A Domain Adaptable Method for Knowledge Tracing // Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining WSDM ’22. New York, NY, USA: Association for Computing Machinery, 2022. С. 123–131.

Подписано в печать 20.06.2023 г.
Формат 60x90/8
Печать цифровая. Печатных листов 9,25
Тираж 100 экз. Заказ № 403

Отпечатано в ФГБУ «Издательство «Наука»
(Типография «Наука»)
121099, Москва, Шубинский пер., 6