

ACTIVE KNOWLEDGE BASE PROTOTYPE ON THE BASIS OF COMPUTATIONAL MODELS

A. Artiukhov

Novosibirsk State University,
630090, Novosibirsk, Russian Federation

DOI: 10.24412/2073-0667-2021-4-55-66

The ever-growing volume of knowledge, for example, in the programming field, requires a person to increase the speed of knowledge processing, to speed up knowledge mastering, and to use it more efficiently. One possible solution is to automate the process of knowledge application. But in many spheres of activity today humanity is accumulating knowledge in an informal way by using linguistic systems. Mastering the reading, understanding and correct use of the knowledge presented in this form, requires a long learning curve. Thus, knowledge accumulated in this form cannot be applied directly and automatically. Moreover, if the person who has mastered it does not use it very often, over time it will begin to be forgotten and subsequently can be completely lost.

In this work, the application of knowledge is considered as creation of a computer program that utilizes it, and the automation of the knowledge application is understood as the automatic synthesis of such programs.

To solve mentioned problems efficiently the system must not only store the knowledge itself, but also save the functional connections between individual concepts. In addition, it is necessary that the system, according to the task specification, is able to automatically construct a software application that solves it.

The system that allows one to automatically apply knowledge is called the Active Knowledge Base. This system is based on the theory of structural synthesis of programs. Knowledge in such a system is represented in the form of computational models, i.e. bipartite graphs in which the vertices in one set correspond to operations, and in the other - to variables. The edges of the graph determine whether a variable is the result of an operation or its input. Variables in such models correspond to some values of the described subject area, and operations are associated with certain program modules at the execution stage. The input and output parameters of program modules are associated with the variables of the computational model.

One of the key steps in constructing a software application is building a computation plan based on the specification of the problem. A computation plan is a partially ordered set of operations, where the order relation is consistent with information dependencies and is defined as „to compute F_2 , you need to compute F_1 “.

Within this prototype, knowledge in the form of computational models is stored in a special Sqlite database, and the system itself is divided into 3 parts. The first part is a subsystem for storing computational models. It adds new computational models to the database and reads them from it. The second part is a subsystem for constructing computation plans, which is engaged in building computation plans according to the specification of the problem. The third part is an execution subsystem. The computation plan is not a complete program that could be executed by the user's OS. To execute it, one needs a system that interprets the plan. In this work it's done by the execution subsystem.

Computational models are described in special files, where each file is a description of one entity, for example, a variable of a computational model or an operation.

To construct a computation plan, a problem specification should be described first. It consists of the name of the computational model, a set of input variables, and a set of output variables. When the specification is completed, the file describing it is fed to the subsystem for constructing computation plans via command line arguments. Having received it, the system launches the construction algorithm, its result is a file with a computation plan.

The general operating principle of the execution subsystem is similar to that of the interpreter. Operations are executed sequentially, as the variables on which they depend are ready. The execution subsystem supports the initialization of the input variables of the computation plan with the initial values passed to it; for this, it needs to create a special file with initialization parameters and send it to the execution subsystem via command line.

The process of operations execution continues until one of the conditions is met: all operations from the computation plan have already been calculated, or all output variables of the computation plan have been calculated. If in the course of calculations all operations from the calculation plan have been executed, and not all of the output variables have been calculated, then the calculations are completed with a corresponding warning message sent to the user. Otherwise, when the variables have already been calculated, and there are still operations in the computation plan, then the plan execution ends normally, the output variables are saved in accordance with their types, and the user is informed about the result of the computations through the standard output stream.

Results of the development and implementation of the system called Active Knowledge Base are presented in the paper.

Key words: Active knowledge, Program synthesis, Active knowledge base, Structural synthesis of programs, Automatic programs construction, Knowledge base, Knowledge storage, Computational models.

References

1. Zagorulko YU. A., Borovikova O. I. Podhod k postroeniyu portalov nauchnyh znaniy // *Avtometriya*. 2008. V. 44. № 1. P. 100–110.
2. Sokolova E. G., Kononenko I. S., Zagorulko YU. A. Problemy opisaniya komp'yuternoj lingvistiki v vide ontologii dlya portala znaniy // *Komp'yuternaya lingvistika i intellektual'nye tekhnologii: Trudy mezhdunarodnoj konferencii „Dialog 2008“ (Bekasovo, 4–8 iyunya 2008 y.)*. M.: RGGU, 2008. N 7 (14), P. 482–487.
3. Borovikova O. I. et al. Razrabotka portala znaniy po komp'yuternoj lingvistike // *KII–2008*. 2008. P. 380–388.
4. Gennari J. H. The evolution of Protégé: an environment for knowledge-based systems development / J. H. Gennari, M. A. Musen, R. W. Ferguson, W. E. Grosso, M. Crubčzy, H. Eriksson, N. F. Noy, S. W. Tu // *International Journal of Human-Computer Studies*. 2003. V. 58, N 1. P. 89–123.
5. McGuinness D. L. et al. OWL web ontology language overview // *W3C recommendation*. 2004. V. 10. N 10. P. 2004.
6. Kurbatov S. S., Lobzin A. P., Hahalin G. K. Instrumental'nye sredstva postroeniya ontologii dlya sinteza programm // *Naukoemkie tekhnologii*. 2014. V. 15. N 1. P. 098–100.
7. Loveland D. W. Automated Theorem Proving: A Logical Basis / D. W. Loveland. — Amsterdam : North Holland, 1978. 418 p. *Fundamental Studies in Computer Science*.
8. Rautiajnien A. Avtomaticheskaya generaciya logicheskogo znaniya // *Discrete and Continuous Models and Applied Computational Science*. 2008. N 4.
9. Mazakov E. B. Predstavlenie i obrabotka znaniy v gibridnyh informacionnyh avtomatizirovannyh sistemah // *Innovacii v nauke*. 2013. N 24.

-
10. Pavlov V. A., Pak V. G. Sistema avtomaticheskogo dokazatel'stva teorem intuicionistskoj logiki na osnove obratnogo metoda // *Programmirovanie*. 2018. N 1. P. 46–59.
 11. Davydov A. V., Larionov A. A., Cherkashin E. A. Ob ischislenii pozitivno-obrazovannyh formul dlya avtomaticheskogo dokazatel'stva teorem // *Modelirovanie i analiz informacionnyh sistem*. 2010. V. 17. N 4. P. 60–70.
 12. Bratko I. *Prolog Programming for Artificial Intelligence* / I. Bratko. — Harlow : Pearson Education, 2001.
 13. Valkovskij V. A. Sintez parallel'nyh programm i sistem na vychislitel'nyh modelyah : monografiya / V. A. Val'kovskij, V. E. Malyshkin. — Novosibirsk: Izd-vo „Nauka“ Sibirskoe otd-e, 1988.
 14. Malyshkin V. E. Strukturnyj sintez parallel'nyh programm / V. E. Malyshkin // *Sbornik tret'ej shkoly-seminara po parallel'nym i vysokoproizvoditel'nym vychisleniyam: sb. statej*. Tomsk, 2005. P. 3–9.
 15. Artiukhov A. A., Parfenov D. R. Razrabotka i realizaciya podsistemy hraneniya i primeneniya programmnyh modulej dlya bazy aktivnyh znaniy // *Informacionnye tekhnologii*. 2019. P. 155.

ПРОТОТИП БАЗЫ АКТИВНЫХ ЗНАНИЙ НА ОСНОВЕ ВЫЧИСЛИТЕЛЬНЫХ МОДЕЛЕЙ

А. А. Артюхов

Новосибирский национальный исследовательский государственный университет,
630090, Новосибирск, Россия

УДК 519.685.7+ 004.4'236

DOI: 10.24412/2073-0667-2021-4-55-66

В статье представлен результат разработки системы, которая позволяет сохранять информацию о программных модулях и связях между ними в форме, подходящей для автоматического применения при конструировании прикладных программ. В основе системы лежит теория структурного синтеза программ. Знания в такой системе представляются в виде вычислительных моделей, а в качестве атомарной единицы знаний рассматривается программный модуль. В результате проведенного исследования была разработана архитектура системы, разработан алгоритм автоматического конструирования прикладных программ и проверена работа системы на ряде тестов.

Ключевые слова: структурный синтез программ, активные знания, синтез программ, база активных знаний, автоматическое конструирование программ, база знаний, хранение знаний, вычислительные модели.

Введение. Во многих сферах деятельности сегодня человечество накапливает знания в неформальном виде при помощи языковых систем. Однако, чтобы освоить чтение, понимание и корректное использование знаний, представленных в таком виде, требуется длительное обучение. Центральная проблема такого подхода — знания, накопленные в таком виде, не могут быть применены непосредственно и автоматически. Более того, если освоивший их человек не будет ими достаточно часто пользоваться, то со временем они начнут забываться и впоследствии могут быть полностью утрачены.

С другой стороны, когда некоторое прикладное решение уже создано, то зачастую неизвестно, какие именно знания были использованы для его получения. Если знания, использованные при создании такого решения, расширятся, то автоматически изменить это решение с учетом новых знаний будет уже затруднительно.

Задача автоматизации работы со знаниями особенно актуальна в сфере программирования. Например, при разработке новых прикладных решений часто требуется найти подходящий программный модуль среди имеющихся и применить его. Количество доступных для программиста модулей растет с каждым днем, как и их сложность. С ростом сложности решаемой прикладной задачи обычно увеличивается количество задействованных в ее решении модулей, а следовательно, затрудняется работа с ними в ручном режиме.

В рамках статьи под применением знаний для решения некоторой задачи понимается конструирование прикладной программы, решающей поставленную задачу. К решению задач по автоматизации работы со знаниями сложилось три основных подхода: онтологический подход, логический синтез программ, а также структурный синтез программ.

Онтологический подход [1] подразумевает формализацию знаний в виде онтологий, инструмента, подходящего для различных предметных областей, например, компьютерной лингвистики [2, 3]. Для построения онтологий активно развиваются такие программные средства, как Protégé [4] и OWL [5]. Однако в общей постановке онтологический подход не ориентирован на автоматизацию применения знаний. Существенные трудности при реализации системы, нацеленной на решение этой задачи, создаст то, что не по всякой онтологии возможно автоматическое построение прикладной программы [6].

Логический синтез программ [7] подразумевает формализацию знаний в виде аксиоматических теорий. Возможность автоматической обработки знаний, представленных в таком виде, уже была показана ранее [8, 9]. Однако описание какой-либо нетривиальной предметной области в виде аксиоматической теории — трудно реализуемая задача. Для конструирования прикладной программы этот подход опирается на методы автоматического доказательства теорем [10, 11]. В этой сфере создано множество языков программирования, например, таких как Prolog [12], и подход продолжает активно исследоваться.

Структурный синтез программ [13] предлагает формализовать знания в виде вычислительных моделей. Одно из преимуществ такого подхода в том, что конструирование некоторого прикладного решения по вычислительной модели — относительно простая задача. Также, в структурном синтезе программ заложена возможность оптимизации создаваемого прикладного решения [14] на уровне выбора знаний, используемых при его конструировании. Применимость структурного синтеза программ для автоматизации работы со знаниями уже исследовалась в предыдущей работе автора [15].

Наиболее удобный подход для решения ранее описанных проблем — структурный синтез программ. С точки зрения проблем автоматизации работы со знаниями этот подход наиболее целостный и ориентированный на них. Чтобы эффективно решать обозначенные проблемы, система должна не только хранить сами знания, но и функциональные связи между ними. Основные требования к такой системе — это возможность хранения формализованных знаний и возможность автоматического конструирования прикладных программ на их основе.

В статье представлены результаты разработки и реализации системы, называемой База Активных Знаний (БАЗ), которая позволяет сохранять информацию о программных модулях и связях между ними в форме, подходящей для автоматического конструирования новых прикладных программ. Синтез таких программ по их спецификации происходит автоматически.

1. Термины и определения. Проблемы формализации и автоматизации применения знаний в рамках структурного синтеза программ решаются с помощью сохранения знаний о программных модулях и способах их применения в виде вычислительных моделей. Вычислительная модель — это двудольный ориентированный граф, где вершины в одной доле соответствуют операциям, а в другой — переменным. Дуга графа, соединяющая операцию с переменной, определяет переменную как результат этой операции, а противоположно направленная дуга определяет переменную как входной аргумент. Переменные в модели соответствуют каким-либо величинам описываемой предметной области, а операциям на этапе исполнения сопоставляется некоторый программный модуль. Входным и выходным параметрам модуля при этом ставятся в соответствие переменные вычислительной модели.

Один из ключевых этапов по созданию прикладной программы — построение плана вычислений по спецификации задачи. План вычислений — это частично упорядоченное

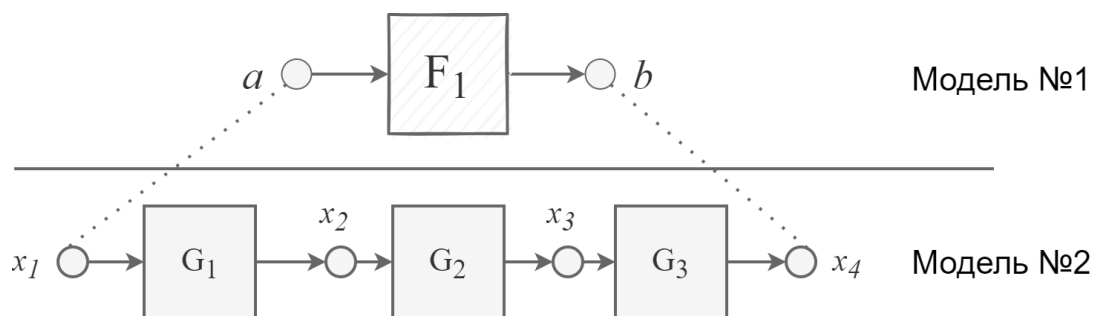


Рис. 1. Структурированная операция F_1 , ссылающаяся на вычислительную модель № 2

множество операций, где отношение порядка согласовано с информационными зависимостями и определяется как „для вычисления F_2 требуется вычислить F_1 “. Спецификация задачи состоит из вычислительной модели, на которой она ставится, множества входных переменных для прикладной программы и множества переменных, которые необходимо вычислить.

По одной и той же спецификации может быть построено несколько различных по нефункциональным свойствам планов вычислений, при этом не всякий из этих планов будет удовлетворителен с точки зрения практического использования. В этом случае появляется необходимость упорядочивания построенных планов вычислений с точки зрения их практичности. Для этого ставится оптимизационная задача и вводится понятие критерия оптимизации, по которому и будет осуществляться такое упорядочивание. Основой для критерия оптимизации могут быть, например, нефункциональные свойства модулей.

Если у операции определены какие-либо входные переменные, то будем говорить, что имеет место информационная зависимость операции от этих переменных. А если переменная является выходной переменной какой-либо операции, то будем говорить, что имеет место информационная зависимость переменной от этой операции. Информационная зависимость операции от переменной считается разрешенной, если у этой переменной уже имеется какое-либо значение. Зависимость переменной от операции разрешается путем исполнения этой операции.

Примеры вычислительных моделей показаны на рис. 1 и рис. 2. Квадратами на них изображаются операции, а кругами переменные. Стрелками показаны информационные зависимости. Для применения знаний из одной вычислительной модели в рамках другой модели в структурном синтезе применяются структурированные операции. Например, на рис. 1 операция F_1 является структурированной и ссылается на модель № 2. Условные переходы внутри вычислительной модели описываются операциями ветвления (рис. 2).

2. Прототип базы активных знаний. В рамках исследования структурного синтеза программ в лаборатории синтеза параллельных программ ИВМ и МГ СО РАН был разработан прототип базы активных знаний для операционных систем из семейств Linux и Windows. В рамках этого прототипа, знания в виде вычислительных моделей сохраняются в специальную Sqlite базу данных, а сама система разбита на 3 части. Первая часть — подсистема хранения вычислительных моделей, она осуществляет добавление новых вычислительных моделей в базу данных и чтение моделей из нее. Вторая часть — подсистема построения планов вычислений, которая занимается построением планов вычислений по спецификации задачи. Третья часть — подсистема исполнения планов вычислений. План вычислений не является полноценной программой, которую бы могла исполнить ОС поль-

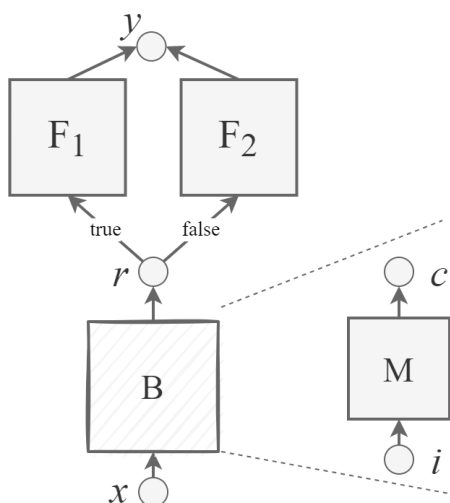


Рис. 2. Операция ветвления В ссылается на модель, реализующую вычисление условия ветвления

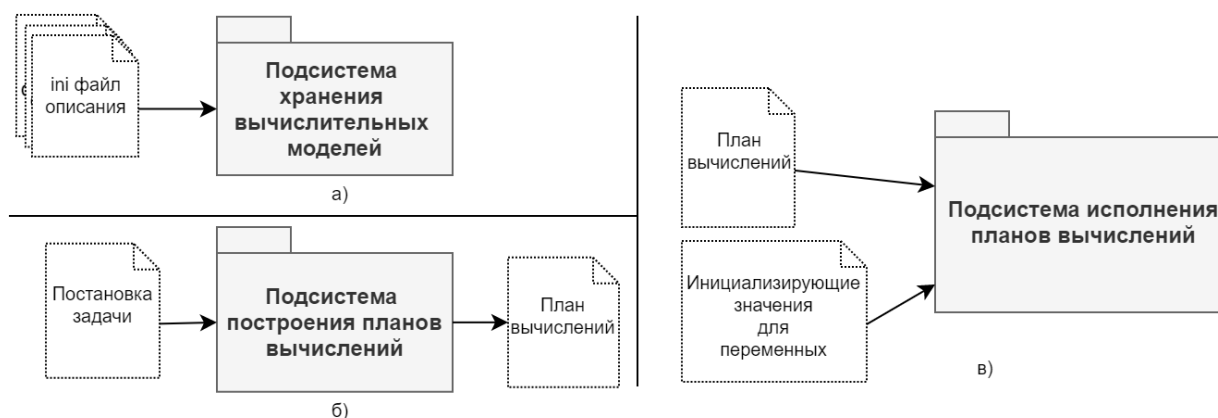


Рис. 3. Схема взаимодействия с подсистемой хранения (а), подсистемой построения планов (б), подсистемой исполнения (в)

зователя. Для его исполнения необходима система, которая его интерпретирует. В БАЗе этим занимается подсистема исполнения. Схема взаимодействия с системой представлена на рис. 3.

Для описания вычислительных моделей используется специальный формат, разработанный автором в рамках предыдущих исследований структурного синтеза программ [15]. В нем описание происходит путем составления ini-файлов, где каждый файл — это какая-то сущность, например, переменная вычислительной модели или операция.

Вычислительная модель описывается уникальным названием и ее описанием. Переменная, помимо названия и описания, также имеет тип, определяющий способ ее хранения, и относится к некоторой вычислительной модели. Операция имеет название, описание, тип, привязывается к вычислительной модели, но кроме этих параметров у операции также задаются ее входные и выходные аргументы. У операций и переменных также могут быть дополнительные параметры. Например, у некоторых типов операций задается дополни-

тельный параметр, определяющий, как будет запускаться программный модуль. Один из таких типов операций — команда командной строки операционной системы.

Описание структурированных операций содержит такие параметры, как имя вложенной вычислительной модели и привязка переменных внешней вычислительной модели к переменным вложенной. Операции ветвления описываются похожим образом, но дополнительно необходимо указывать, вычисление каких операций должно последовать в зависимости от результата условия.

Подсистема построения планов вычислений отвечает за их конструирование. Для начала нужно составить спецификацию задачи, которая состоит из имени вычислительной модели, множества входных переменных и множества выходных переменных, причем все переменные должны принадлежать той вычислительной модели, на которой ставится задача. Когда спецификация составлена, файл, ее описывающий, подается в подсистему построения через аргументы командной строки. Получив его, система запускает алгоритм построения плана вычислений, результат работы которого — файл с планом вычислений.

Алгоритм построения планов вычислений состоит из четырех шагов.

На первом шаге строится граф операций, для которых либо уже разрешены информационные зависимости, либо могут быть разрешены путем исполнения операций из этого графа.

На втором шаге происходит отбрасывание операций, не задействованных в вычислении выходных переменных задачи. Для этого применяется обход графа операций с предыдущего шага, однако теперь он идет в сторону против информационных зависимостей, а в качестве стартовых вершин берутся выходные переменные задачи.

Третий шаг осуществляет планирование для структурированных операций и операций ветвления.

На последнем шаге для каждой выходной переменной из спецификации строится последовательность операций, которые необходимо вычислить, чтобы получить эту переменную. Если для переменной в этой модели существует несколько альтернативных последовательностей операций, ее вычисляющих, то среди них выбирается достаточно хорошая последовательность с точки зрения используемого критерия. Используемый в алгоритме критерий можно изменить, но так как он является неотъемлемой частью алгоритма построения планов вычислений, то после его изменения требуется перекомпиляция этой подсистемы.

Когда вычисление каждой выходной переменной было спланировано, работа алгоритма останавливается, возвращая построенный план вычислений.

Общий принцип работы подсистемы исполнения схож с работой интерпретатора. Исполнение операций происходит последовательно, по мере готовности переменных, от которых они зависят. Подсистема исполнения поддерживает инициализацию входных переменных плана вычислений переданными ей начальными значениями, для этого ей необходимо описать и передать специальный файл с параметрами инициализации.

В зависимости от типа переменной ее значение может храниться в различных местах, например, в файле. Поэтому перед исполнением операции сначала происходит загрузка переменных, связанных с ее входными аргументами, а также подготовка выходных аргументов к приему данных, если таковая необходима. После этого происходит вычисление самой операции, и подсистема сохраняет ее выходные аргументы в соответствующие переменные.

Исполнение структурированных операций происходит иначе. Вычислительную модель, на которую ссылается структурированная операция, будем называть вложенной моделью. После загрузки входных аргументов их значения копируются в переменные вложенной модели, указанной в этой операции, а затем уже запускается план вычислений для этой модели. При завершении исполнения плана вычислений значения переменных вложенной модели копируются в переменные основной модели. Исполнение операции ветвления отличается от исполнения структурированных операций тем, что, после вычисления результата ее условия, подсистема исполнения в качестве следующей операции к исполнению берет ту, которая соответствует этому результату.

Процесс исполнения операций продолжается, пока не будет выполнено одно из условий: уже вычислены все операции из плана вычислений, все выходные переменные плана вычислений вычислены. Если в процессе вычислений были исполнены все операции из плана вычислений, а выходные переменные вычислены не все, то вычисления завершаются с отправкой соответствующего предупреждающего сообщения пользователю. В противном случае, когда переменные уже вычислены, а в плане вычислений еще есть операции, то исполнение плана завершается штатно, выходные переменные сохраняются в соответствии с их типами, а в поток стандартного вывода пользователю сообщается о результате вычислений.

Разработанная система позволяет осуществлять хранение и автоматическое применение знаний, формализованных в виде вычислительных моделей со структурированными операциями и операциями ветвления.

3. Тестирование. Для проверки работы разработанной системы с операциями, ссылающимися на другие вычислительные модели, было разработано два набора тестов.

Первый набор тестов подразумевает описание некоторой задачи в виде вычислительных моделей со структурированными операциями, синтез по ним прикладной программы, и, после этого, исполнение полученного плана для соотнесения результата вычислений с ожидаемым результатом. Для этого теста были разработаны три вычислительных модели: модель № 1, № 2 и № 3. Модель № 1 состоит только из одной операции, которая является структурированной операцией и реализуется моделью № 2. Модель № 2 состоит из трех операций, причем операция G2 структурированная и реализуется моделью № 3. Полная схема связей между вычислительными моделями представлена на рис. 4. Модели были описаны и переданы в подсистему хранения. Затем была описана задача вычисления переменной b из переменной a и передана в БАЗу. Полученный план вычислений был передан в подсистему исполнения.

Наблюдаемый при исполнении плана вычислений порядок вызова операций представлен на рис. 5 и соответствует требуемому порядку.

Второй набор тестов подразумевает описание некоторой задачи в виде вычислительных моделей с операциями ветвления и исполнение плана вычислений для этой задачи. При этом нужно убедиться, что порядок вычислений совпадает с ожидаемым. Для этого на последнем шаге тестирования исполнение программы нужно произвести два раза по разным путям в вычислительной модели.

В рамках теста была разработана основная вычислительная модель, названная Модель № 1 на рис. 6, а также вспомогательная вычислительная модель, названная Модель № 2 на рис. 6. Основная модель содержит операцию ветвления V и переменную r , от которой зависит выбор пути вычислений. Условие ветвления в операции V реализуется вспомогательной моделью и состоит из логического умножения его аргументов. Полная схема

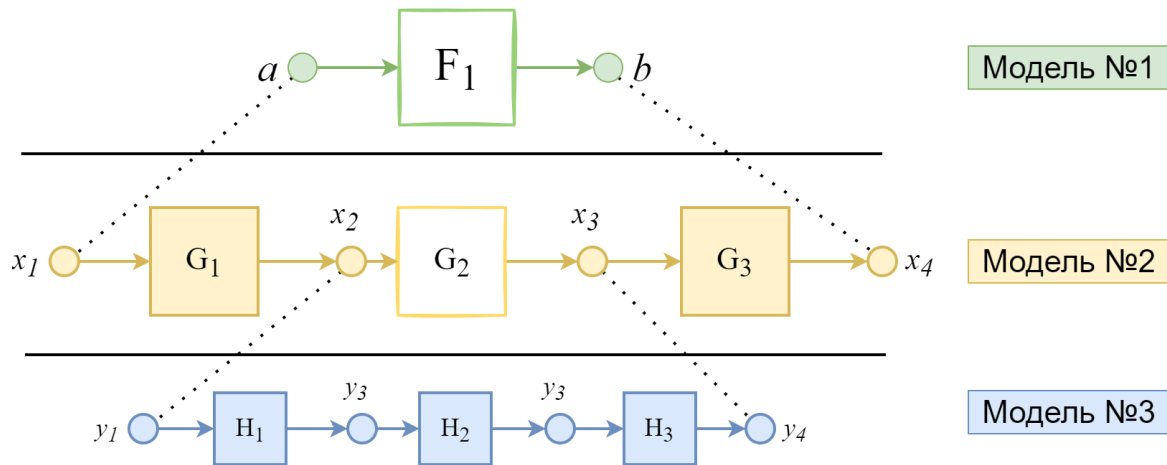


Рис. 4. Вычислительные модели для тестирования структурированных операций

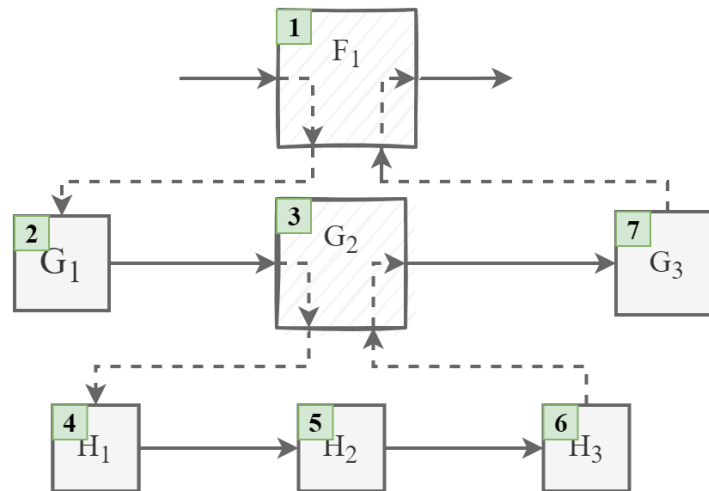


Рис. 5. Порядок вызова операций в тесте № 1

связей между вычислительными моделями представлена на рис. 6. Модели были описаны и переданы в подсистему хранения. Затем была описана и передана в БАЗу задача вычисления переменной y из переменных x_1 и x_2 . Полученный план вычислений был передан в подсистему исполнения.

Наблюдаемый при исполнении плана вычислений порядок вызова операций представлен на рис. 7 и соответствует требуемому порядку.

Заключение. В статье представлена система хранения знаний в формальном виде и их автоматического применения. Разработана система, сохраняющая информацию о подпрограммах и о связях между ними в форме, подходящей для автоматического конструирования новых прикладных программ. Разработанная система позволяет производить автоматический синтез прикладных программ по спецификации задачи, а также поддерживает применение знаний из одной вычислительной модели в рамках другой модели. Разработана возможность описания условных переходов в рамках вычислительной модели. Реализована система для накопления и автоматического применения знаний в виде

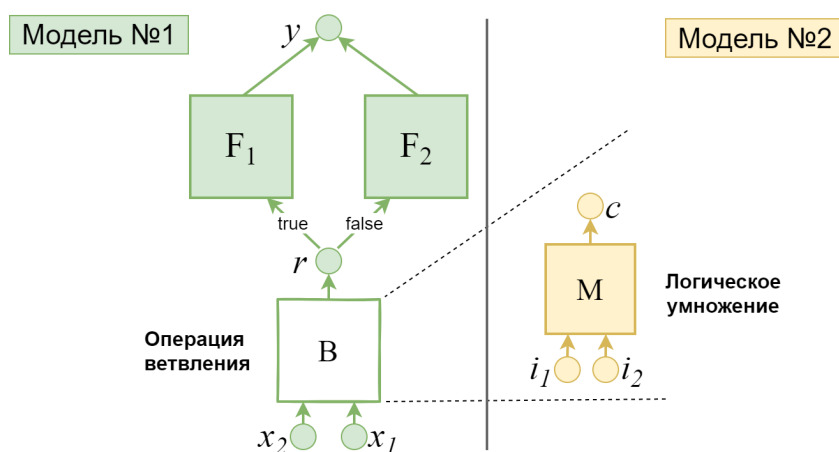


Рис. 6. Вычислительные модели для тестирования операций ветвления

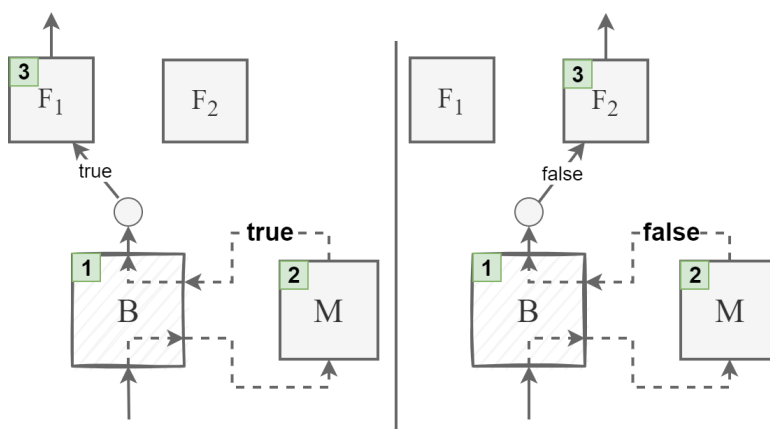


Рис. 7. Порядок вызова операций в тесте № 2

вычислительных моделей, и было проведено ее тестирование на различных вычислительных моделях, что показало применимость подхода и системы на ряде тестов.

В дальнейших исследованиях планируется добавление поддержки других видов вычислительных моделей: рекурсивных вычислительных моделей, вычислительных моделей с массивами, а также вычислительных моделей с очередями.

Список литературы

1. Загоруйко Ю. А., Боровикова О. И. Подход к построению порталов научных знаний // Автометрия. 2008. Т. 44. № 1. С. 100–110.
2. Соколова Е. Г., Кононенко И. С., Загоруйко Ю. А. Проблемы описания компьютерной лингвистики в виде онтологии для портала знаний // Компьютерная лингвистика и интеллектуальные технологии: Труды международной конференции „Диалог 2008“ (Бекасово, 4–8 июня 2008 г.). М.: РГГУ, 2008. Вып. 7 (14), С. 482–487.
3. Боровикова О. И. и др. Разработка портала знаний по компьютерной лингвистике // КИИ-2008. 2008. С. 380–388.
4. Gennari J. H. The evolution of Protégé: an environment for knowledge-based systems development / J. H. Gennari, M. A. Musen, R. W. Ferguson, W. E. Grosso, M. Crubizy, H. Eriksson,

- N. F. Noy, S. W. Tu // International Journal of Human-Computer Studies. 2003. Т. 58, Вып. 1. С. 89–123.
5. McGuinness D. L. et al. OWL web ontology language overview // W3C recommendation. 2004. Т. 10. № 10. С. 2004.
 6. Курбатов С. С., Лобзин А. П., Хахалин Г. К. Инструментальные средства построения онтологии для синтеза программ // Наукоемкие технологии. 2014. Т. 15. № 1. С. 098–100.
 7. Loveland D.W. Automated Theorem Proving: A Logical Basis / D. W. Loveland. — Amsterdam : North Holland, 1978. Fundamental Studies in Computer Science.
 8. Раутиайнен А. Автоматическая генерация логического знания // Discrete and Continuous Models and Applied Computational Science. 2008. № 4.
 9. Мазиков Е. Б. Представление и обработка знаний в гибридных информационных автоматизированных системах // Инновации в науке. 2013. № 24.
 10. Павлов В. А., Пак В. Г. Система автоматического доказательства теорем интуиционистской логики на основе обратного метода // Программирование. 2018. № 1. С. 46–59.
 11. Давыдов А. В., Ларионов А. А., Черкашин Е. А. Об исчислении позитивно-образованных формул для автоматического доказательства теорем // Моделирование и анализ информационных систем. 2010. Т. 17. № 4. С. 60–70.
 12. Bratko I. Prolog Programming for Artificial Intelligence / I. Bratko. — Harlow : Pearson Education, 2001.
 13. Вальковский В. А. Синтез параллельных программ и систем на вычислительных моделях: монография / В. А. Вальковский, В. Э. Малышкин. Новосибирск: Изд-во „Наука“ Сибирское отделение, 1988.
 14. Малышкин В. Э. Структурный синтез параллельных программ / В. Э. Малышкин // Сборник третьей школы-семинара по параллельным и высокопроизводительным вычислениям: сб. статей. Томск, 2005. С. 3–9.
 15. Артюхов А. А., Парфенов Д. Р. Разработка и реализация подсистемы хранения и применения программных модулей для базы активных знаний // Информационные технологии. 2019. С. 155–155.



Артюхов Алексей Андреевич — магистрант факультета информационных технологий, Новосибирский национальный исследовательский государственный университет, 630090, Новосибирск, Россия; e-mail: gridbugkiller@gmail.com.

Артюхов Алексей Андреевич получил степень бакалавра в 2019 году в Новосибирском национальном исследовательском государственном университете на кафедре параллельных вычислений факультета информационных технологий, тема выпускной квалификационной работы — „Разработка и реализация системы хранения и применения программных модулей для базы активных знаний“. Научные инте-

ресы — разработка систем хранения и автоматического применения знаний.

Artyukhov Alexey Andreevich — master's student of the Faculty of Information Technologies, Novosibirsk State University, 630090, Novosibirsk, Russia; e-mail: gridbugkiller@gmail.com.

Artyukhov Alexey Andreevich received a bachelor's degree in 2019 from Novosibirsk State University at the Department of Parallel Computing of the Faculty of Information Technologies, the topic of his qualification work is „Development of a software modules storage and application system for an active knowledge base“. He is interested in development of knowledge storage and automatic knowledge application systems.

Дата поступления — 11.06.2021