Статья опубликована в журнале «Вестник компьютерных и информационных технологий». 2011. № 1, с.38 – 43.

УДК 519.68

Бобровских А.С.

Ярославский Государственный Университет имени П. Г. Демидова, Ярославль

Разработка обучающей системы для алгоритмов начертательной геометрии с использованием технологии автоматного программирования

Введение

В данной работе предлагается подход к созданию систем, обучающих алгоритмам начертательной геометрии на основе технологии автоматного программирования. Разработан язык описания таких систем, позволяющий записывать их логику и поддерживающий основные функциональные возможности, такие как вызов процедур, обработка циклов, использование операторов ветвления, и добавление комментариев.

Предложена **модификация технологии автоматного программирования** для формального преобразования программы на таком языке в систему конечных взаимодействующих автоматов и выделенную модель данных. Такое преобразование позволяет использовать существующие методы верификации, предоставляет возможность трассировки программы, как в прямом, так и в обратном направлении, а также позволяет упростить процесс разработки и расширения таких систем.

Использование программирования с явным выделением состояний [1] обосновано, так как оно существенно облегчает понимание программы. Данная технология в настоящее время развивается и **имеет все шансы стать основной технологией, покрывающей разрыв между идейной разработкой проекта и его реализацией**.

1. Формулировка задачи

В настоящее время существует множество обучающих систем в самых различных предметных областях. Актуальна и проблема обучения алгоритмам начертательной геометрии. В связи с этим было предложено разработать систему, предоставляющую возможность разработки визуального представления и обучения данным алгоритмам.

При изучении начертательной геометрии важную роль играют **визуализаторы алгоритмов**, позволяющие динамически отображать детали их работы. Использование в учебном процессе визуализаторов и систем автоматического контроля открывает возможность использования нового подхода к обучению начертательной геометрии.

Под визуализатором понимается программа, в процессе работы которой на экране компьютера динамически демонстрируется применение алгоритма к выбранному набору данных. Визуализаторы позволяют изучать работу алгоритмов, как в автоматическом, так и в пошаговом режиме, аналогичном режиму трассировки программ. При этом трассировка может осуществляться шагами с различным масштабом, как в прямом, так и в обратном направлении.

Под системой автоматического контроля будем понимать программу, которая отслеживает действия пользователя и сравнивает их с эталонной последовательностью. На основании анализа она (в зависимости от выбранного режима) может либо давать указания по решению задачи, либо записывать ошибки в лог-файл, с последующим подведением итогов решения задачи, выдачей статистических данных, рекомендаций и выставлением оценки.

На рис. 1 в общем виде представлена схема взаимодействия основных модулей. Обучающая система состоит из двух основных частей: системы визуализации и системы обучения. Система визуализации взаимодействует с интерфейсом алгоритма непосредственно, а система обучения – через анализатор.

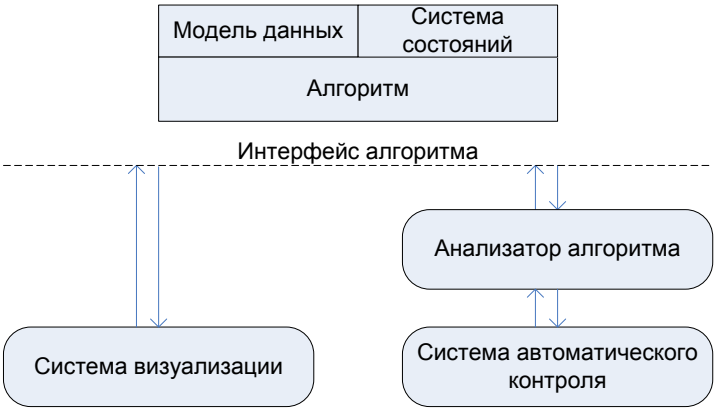

Рис. 1. Основные модули обучающей системы

Визуализатор должен предоставлять возможность выбора данных для инициализации алгоритма, навигацию по шагам, а также отображение комментариев.

Система автоматического контроля должна предоставлять пользователю инструменты, необходимые для решения задачи, а также, если это необходимо, снабжать его указаниями и выдавать данные о результатах его действий.

Для реализации работы визуализатора и системы контроля алгоритм должен предоставлять определенный интерфейс. Этот интерфейс в общем виде представлен на рис. 2.

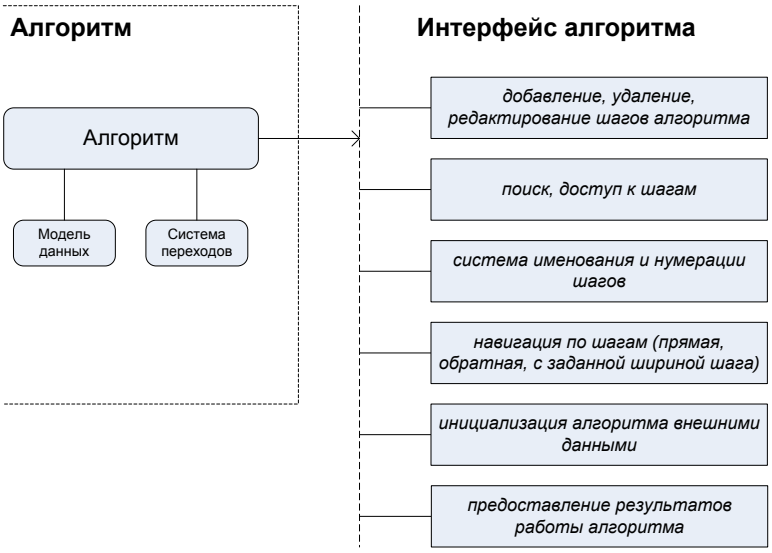

Рис. 2. Интерфейс алгоритма

2. Автоматизация построения обучающей системы

Алгоритмы начертательной геометрии представляют собой последовательность построений различных геометрических объектов – точек, линий, кривых. Каждый такой объект рассматривается как отдельный элемент, который не изменяется. Единственное действие, которое можно с ним произвести – удалить (стереть). Поворот линии, например, также можно рассматривать как удаление существующего элемента и добавление нового.

Число действий в таких алгоритмах относительно невелико по сравнению с алгоритмами дискретной математики. Многие из действий являются неявными, причиной чему служит недостаточная формализация алгоритмов начертательной геометрии. При попытке дать четкую формулировку часто появляется некоторое число дополнительных шагов, которые человек делает автоматически, не задумываясь. Однако для обучающей системы формальное определение таких шагов необходимо.

Рассмотрим алгоритм как выделенную модель данных и систему взаимодействующих автоматов. На рис. 3 представлены описания интерфейсов модели данных и системы переходов, составляющих алгоритм, заданный в такой форме. Вынесение данных алгоритма в отдельную модель означает лишь предоставление ко всем данным (входным, рабочим и выходным) унифицированного доступа. При этом физическое копирование информации не осуществляется.

2.1. Разработка грамматики языка описания алгоритмов начертательной геометрии

Предложим грамматику описания алгоритмов. Заметим, что запись алгоритма должна быть по возможности простой, для того чтобы его написание было возможно не только в среде разработки алгоритмов, но и в любом текстовом редакторе. Пользователь, не обладающий навыками программирования, должен иметь возможность быстро разобраться в грамматике и написать реализацию сложного алгоритма.

Рис. 3. Система переходов и модель данных алгоритма

Также грамматика должна быть по возможности гибкой, для того чтобы при необходимости в нее легко можно было внести соответствующие изменения.

В рамках технологии автоматного программирования для реализации основных действий с алгоритмом (навигации) предлагается строить систему автоматов, соответствующую алгоритму. Представим для начала в общем виде грамматику языка описания алгоритмов:

ALG → INPUTS STEPS
INPUTS → INPUTS INPUT
INPUT → STATEMENT;
STEPS → STEPS STEP_RETURN STEPS | STEPS STEP| STEP
STEP → STEP_ACTION | STEP_FOR | STEP_FOREACH | STEP_IF
STEP_ ACTION → { STATEMENT; }
STEP_FOR → FOR_STATEMENT { STEPS_FOR }
STEP_FOREACH → SET_STATEMENT { STEPS_FOR }
STEP_IF → IF (CONDITION){ STEPS } ELSE { STEPS } | IF (CONDITION){ STEPS }
STEPS_FOR → STEPS | STEPS STEP_BREAK STEPS |

STEPS STEP_CONTINUE STEPS |

STEPS STEP_RETURN STEPS

STEP_BREAK → BREAK_CONDITION;

STEP_ CONTINUE → CONTINUE_CONDITION;

STEP_RETURN → RETURN_CONDITION;

В описании данной грамматики присутствуют как обычные элементы, по которым с помощью предлагаемой технологии будет строиться система взаимодействующих автоматов, так и абстрактные элементы *INPUTS, STEPS,* и *STEP*. Первый представляет собой входные данные алгоритма, вторые два – последовательность шагов. Они соответствует блоку операторов в обычных алгоритмах. Рассмотрим их.

ALG

Описание алгоритма задается элементом *ALG*. Он содержит в себе объявления входных данных и шагов алгоритма. У данного элемента задаются свойства:

- *Name* – для задания имени алгоритма, по которому к нему можно будет обращаться, как к имени шага STEP_ACTION, который будет рассмотрен ниже, и, таким образом, использовать в других алгоритмах;
- *Description* – необязательное свойство, в котором указывается описание алгоритма в целом.

Оба параметра являются строковыми.

INPUT

Элемент *INPUT* представляет собой описание входного параметра алгоритма. Для данного элемента необходимо задать свойства:

- *Name* – имя параметра, по которому к нему будет осуществляться доступ в пределах алгоритма.

- *Type* – тип данных входного параметра. Тип может быть строковым, числовым, или представлять собой один из типов графических элементов, элементарных, либо заданных пользователем.
- *Description* – описание параметра. Данное свойство является необязательным.

STEP_ACTION

Для вызова библиотечных функций рисования графических элементов, произведения измерений, расчета проекций, а также использования ранее написанных пользователем алгоритмов используется элемент *STEP_ACTION*. Данный элемент, по сути, является ключевым. В дереве переходов он всегда является листом.

Данный элемент, как и другие типы шагов, имеет следующие свойства:

- *Name* – имя функции или алгоритма *ALG* с параметрами, соответствующими данной функции или алгоритму.
- *Description* – комментарий к данной функции. Комментарий в зависимости от режима работы обучающей системы может выводиться на экран, поясняя пользователю действия, выполняемые на текущем шаге.

STEP_FOR

Элемент *STEP_FOR* задает цикл, в рамках которого с данными осуществляются однотипные операции.

В начале описания цикла необходимо указывать следующие элементы:

- *Start* – задает начало цикла. В блоке *Start* могут быть описаны начальные значения нескольких элементов. Это могут быть как числовые параметры, например, начальный угол наклона прямой, так и параметры, являющиеся

объектами, как в случае задания начальной точки какого-либо действия. Блок является обязательным;

- *End* – задает крайние значения цикла. Данный блок аналогичен блоку *Start*, за исключением того, что он необязателен. В случае совместного использования блоков *Count* и *Step* данный блок не используется.
- *Count* – блок, задающий количество повторений тела цикла. Данный блок является обязательным, за исключением указания блоков *Start*, *End* и *Step*, и использования лишь одной изменяющейся переменной в цикле.
- *Step* – шаг изменения переменной цикла. Числовой параметр. При отсутствии блока *Count* его использование обязательно.
- *Trace* – траектория изменения параметра цикла. Используется в том случае, если изменяемый параметр – точка, и расположение следующей точки определяется какой либо линией, как прямой, так и кривой. Может принимать значения: 1. *line* – изменение параметра линейно. Это значение по умолчанию. 2. «имя_объекта» – один из объектов модели данных – некая кривая, либо прямая. Если данный параметр задан, то *Start* и *End* обязательно должны принадлежать данной линии.

STEP_FOREACH

Данный элемент задает цикл, в котором над множеством элементов производятся однотипные операции. Для описания множества элементов используется слово *Set*. У данного элемента также указывается свойство *Name* – имя множества.

Описание представляет собой список элементов, указанных через точку с запятой. Это могут быть как элементы входных данных, так и результаты выполнения других шагов алгоритма. Для получения доступа к данным шагов используются стандартные функции интерфейса алгоритма.

STEP_IF

Условный оператор задается с помощью свойств: *Condition, IFTrue, IFFalse.* Они соответственно обозначают условие, начало блока, выполняемого в случае его истинности, и начало блока, выполняемого, если условие неверно.

2.2. Динамическое построение системы взаимодействующих конечных автоматов по описанию алгоритма

Рассмотрим автоматы, соответствующие данной грамматике. Для того, чтобы упростить рисунки и не загромождать их лишними подписями и состояниями, автоматы будем представлять в вольном стиле, достаточном для понимания основной концепции.

На рис. 4 представлен автомат алгоритма. Пунктирная линия означает, что элемент *STEPS* является абстрактным. Он представляет собой последовательность всех шагов алгоритма.

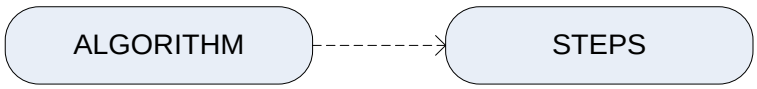

Рис. 4. Упрощенный автомат алгоритма

На рис. 5 представлены возможные варианты последовательности шагов.

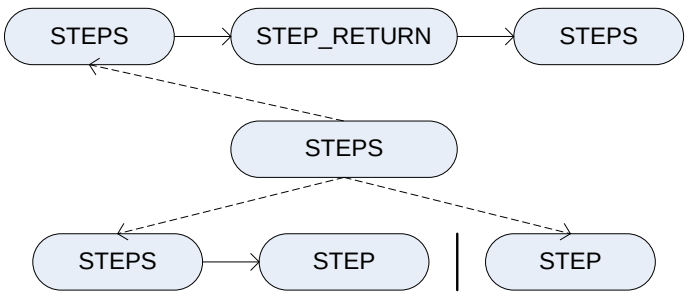

Рис. 5. Структура шагов алгоритма

Каждый шаг алгоритма может быть представлен одним из четырех типов (рис. 6). Данные типы выбраны исходя из описаний алгоритмов начертательной геометрии и отражают основные необходимые варианты действий в рамках таких алгоритмов.

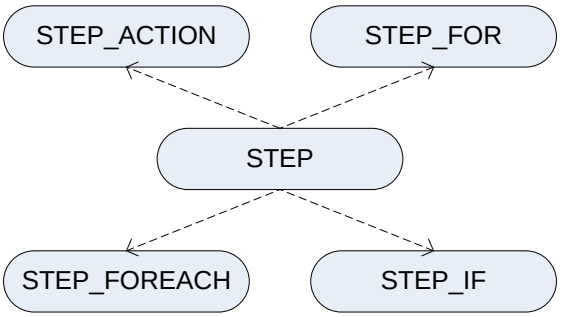

Рис. 6. Виды шагов алгоритма

Каждый из вышеперечисленных видов шагов представляет собой конечный автомат, вызываемый из родительского автомата, которым является либо корневой автомат алгоритма, либо автомат одного из циклов, либо автомат условного оператора.

Автомат *STEP_ACTION* представляет собой в простейшем виде автомат с одним состоянием, в котором происходит выполнение стандартной функции. Оператору цикла со счетчиком соответствует автомат, показанный на рис. 7.

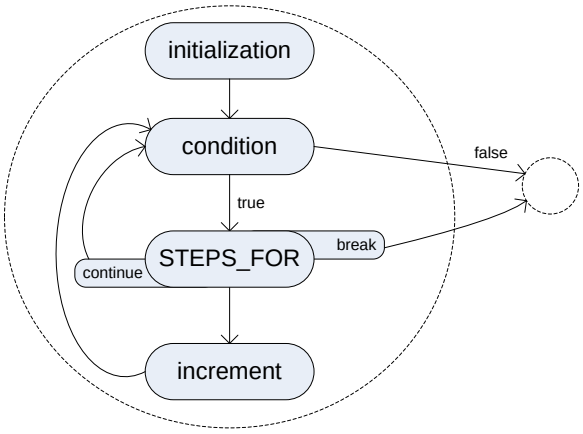

Рис. 7. Автомат оператора цикла со счетчиком

В состоянии инициализации происходит задание начальных параметров, указанных с помощью элементов *Start, End, Step* и *Count*.

Далее следует проверка на выполнение условий: превышение счетчиком максимально допустимого значения, выхода изменяющегося параметра за пределы, указанные при инициализации.

Затем, в случае продолжения работы цикла, осуществляется последовательный вызов вложенных автоматов-шагов. Отдельно выделяются два вида шагов – *STEP_CONTINUE* и *STEP_BREAK*, которые соответственно продолжают работу цикла с начала, либо прерывают ее и переводят автомат цикла в конечное состояние.

Цикл, заданный элементом *STEP_FOREACH*, достаточно схож с предыдущим. В нем вместо начальных условий и счетчика задается множество элементов, над которыми выполняются преобразования. На рис. 8 представлен соответствующий автомат.

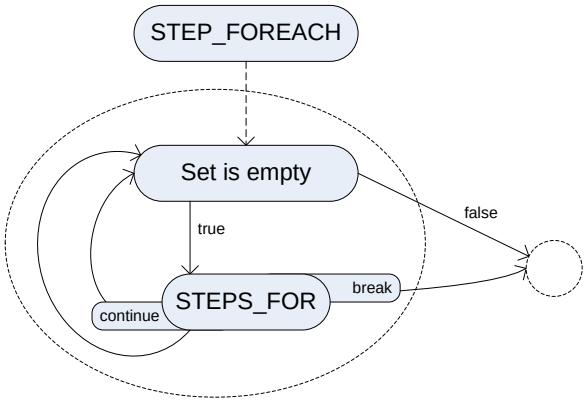

Рис. 8. Автомат оператора цикла *foreach*

Наконец, условный оператор *STEP_IF* представляет собой простой автомат с двумя (либо одной, в случае укороченного оператора ветвления) ветвями, в которых осуществляется последовательный вызов вложенных автоматов. Выбор ветви определяется выполнением условия, заданного элементом *Condition*.

На рис. 9 представлены автоматы полного и укороченного операторов условного перехода. Ветвь вызовов, выполняемая при истинном значении условия, задается элементом *IFTrue*, при ложном значении – элементом *IFFalse*.

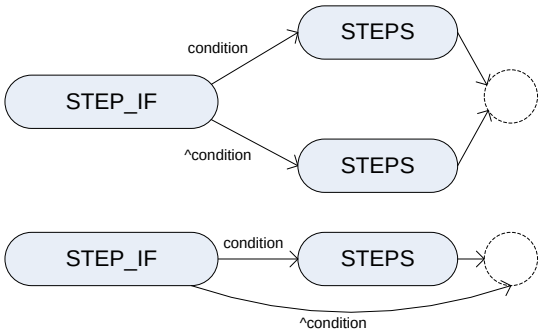

Рис. 9. Автомат оператора условного перехода

Состыковывая фрагменты-автоматы, соответствующие шагам алгоритма, получим иерархию вложенных автоматов. Может показаться, что такое представление искусственно усложнено, и в нем нет необходимости. Однако, данное представление каждого шага алгоритма в виде автомата, вместо состояния автомата, оправдано.

Во-первых, в случае возникновения необходимости добавить схожие состояния во все шаги алгоритма, например, состояние проверки исходных данных для шага или разделение отображения комментариев и непосредственно действий на два отдельных состояния, автомат, в котором каждый шаг представлен состоянием, будет разрастаться, и его понимание станет все более сложной задачей. Если же шаги заданы отдельными автоматами, то основная последовательность не претерпит изменений, последовательность вызовов автоматов не изменится. Подвергнутся модификации лишь сами автоматы шагов. Но поскольку каждый из них достаточно прост – в большинстве случаев это просто линейная система переходов, то их понимание не усложнится. Во-вторых, это необходимо при анализе действий пользователя системой автоматического контроля, так как выполненные шаги алгоритма естественно представлять в виде автоматов, находящихся в конечном состоянии. Если же использовать вместо автоматов обычные состояния, то возникнет необходимость хранить их номера, что вызывает дополнительные расходы и нарушает концепцию *SWITCH*-технологии.

Таким образом, каждый шаг алгоритма представляет собой отдельный экземпляр автомата. В работе [2] предлагается способ формального построения кода программы на

языке *Java* по автомату с помощью шаблонов с последующей проверкой и компиляцией полученной программы. В данной статье предлагается динамическое построение автомата. Похожий способ предложен в работах [3, 4]. Вместо использования переключателя *switch* для хранения состояний и использования процедуры в качестве автомата используются отдельные объекты классов. Каждый такой объект представляет автомат и содержит в себе состояния и функции входных и выходных воздействий как члены класса. Такой подход является расширением подхода, изложенного в работе [2], и также позволяет получить сходные результаты. Достаточно реализовать специальные функции-члены класса, генерирующие код программы, аналогичный коду, генерируемому по шаблонам, как это предлагалось ранее.

Подход, использующий переключатели *switch*, по-прежнему остается актуален, так как он предназначен для программирования контроллеров устройств, в которых использование языков высокого уровня ограничено или невозможно. Тем не менее, во многих других задачах, где применение объектно-ориентированного программирования более чем оправдано, описанный подход, несомненно, удобен, и может быть использован для более широкого класса задач.

Результаты

В данной работе предложен подход к созданию систем, обучающих алгоритмам начертательной геометрии. Также был предложен язык описания таких систем и рассмотрена технология построения системы автоматов по программе на таком языке. Данная технология является модификацией технологии автоматного программирования, которая может быть успешно применена с использованием объектно-ориентированного программирования.

Список литературы

1. *Шалыто А. А.* *SWITCH*-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. 628 с.

2. *Шалыто А. А., Туккель Н. И.* От тьюрингова программирования к автоматному // Мир ПК. 2002. № 2, с. 144 – 149.
3. *Шалыто А. А., Туккель Н. И.* Объектно-ориентированное программирование с явным выделением состояний // Материалы международной научно-технической конференции «Искусственный интеллект – 2002». Т.1. Таганрог – Донецк: ТГРУ – ДИПИИ, 2002, с. 198 – 202.
4. *Шопырин Д. Г., Шалыто А. А.* Применение класса *STATE* в объектно-ориентированном программировании с явным выделением состояний // Труды X Всероссийской научно-методической конференции «Телематика 2003». Т.1. СПбГИТМО (ТУ). 2003, с. 284, 285.