УДК 004.434

АВТОМАТНЫЙ МЕТОД ОПРЕДЕЛЕНИЯ ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ ЯЗЫКОВ (Часть 2)[1]

Ф. А. Новиков,
канд. физ.-мат. наук, заведующий лабораторией астрономического программирования
Институт прикладной астрономии РАН
У. Н. Тихонова,
аспирант
Санкт-Петербургский государственный политехнический университет

Описывается новый метод определения синтаксиса и семантики проблемно-ориентированных языков с помощью диаграмм классов и диаграмм автоматов. Во второй части статьи приводится используемая автоматная модель и рассматривается задание конкретного синтаксиса с помощью системы взаимодействующих автоматов на примерах языка описания шахматных позиций и мини-языка множеств.

Ключевые слова *— проблемно-ориентированный язык, абстрактный синтаксис, метамодель, автоматное программирование.*

Автоматный метод и модель системы автоматов

В первой части статьи рассмотрена центральная составляющая определения проблемно-ориентированного языка — метамодель (в том числе абстрактный синтаксис), задающая структуру языка. Определение структуры проблемно-ориентированного языка необходимо, но не достаточно. Для использования языка, кроме абстрактной структуры, нужны ее конкретная реализация — конкретный синтаксис языка — и способ применения для конкретных целей — семантика языка.

Конкретный синтаксис предоставляет механизм создания (ввода) и изменения (редактирования) программы как абстрактной структуры с помощью ее конкретного представления[2]. Другими словами, конкретный синтаксис определяет однозначное соответствие из представления в экземпляр метамодели. Редактируя представление (текст программы), мы изменяем состав и связи экземпляров классов метамодели. Таким образом, с одной стороны, конкретный синтаксис — это нотация, используемая для изображения (текстового, графического или иного) конструкций языка и их комбинаций. С другой стороны, конкретный синтаксис — это способ редактировать программу и транслировать изображение ее конструкций в набор экземпляров классов метамодели[3]. В автоматном методе определения языков конкретный синтаксис задается как *алгоритм анализа и синтеза* (преобразователь), анализирующий конкретное представление программы с использованием соответствующей нотации и строящий абстрактную программу.

Однако построение экземпляра метамодели редко является самоцелью — как правило, с построенной абстрактной программой нужно еще что-то сделать: сразу выполнить или преобразовать в какой-то другой вид для последующего выполнения. Именно этот результат, т. е. процесс выполнения в конкретной модели вычислимости или исполнимый код в конкретной системе программирования считаются смыслом программы, а соответствие, сопоставляющее программе ее смысл, называется семантикой программы [2]. Существуют различные способы определения семантики языка, среди которых в программировании чаще всего используется операционная семантика. Операционный подход к определению семантики языка предполагает описание *алго-*

[1] Продолжение. Начало в № 6, 2009 г.

[2] А. П. Ершов предложил более точный, но менее распространенный термин «воплощение» [1]. В настоящее время в зарубежной литературе используется термин «reification».

[3] Для краткости набор экземпляров классов метамодели с установленными связями мы называем экземпляром метамодели.

ритма интерпретации программы в терминах некоторой абстрактной машины [3]. В автоматном методе в качестве такой абстрактной машины предлагается использовать систему автоматов.

Таким образом, в автоматном методе конкретный синтаксис описывается в виде алгоритма анализа и синтеза экземпляра метамодели языка, а операционная семантика описывается в виде алгоритма интерпретации этого экземпляра метамодели языка. Причем для описания этих алгоритмов используется одна и та же модель системы взаимодействующих автоматов (которую мы определим ниже в этом разделе). Такая унификация, по нашему мнению, является достоинством автоматного метода.

С точки зрения парадигмы автоматного программирования [4], описание алгоритма в виде системы автоматов и есть реализация этого алгоритма. Мы следуем парадигме автоматного программирования и используем виртуальную машину автоматного программирования, которая интерпретирует автоматы определения проблемно-ориентированного языка и тем самым автоматически реализует анализ и выполнение программ на этом языке.

Основные варианты использования, связанные как с определением, так и с применением проблемно-ориентированных языков, представлены на рис. 9[4]. Здесь система автоматов определения конкретного синтаксиса интерпретируется виртуальной машиной автоматного программирования, и тем самым выполняется анализ программы. Аналогично система автоматов определения семантики интерпретируется той же машиной автоматного программирования, и тем самым выполняется интерпретация программы.

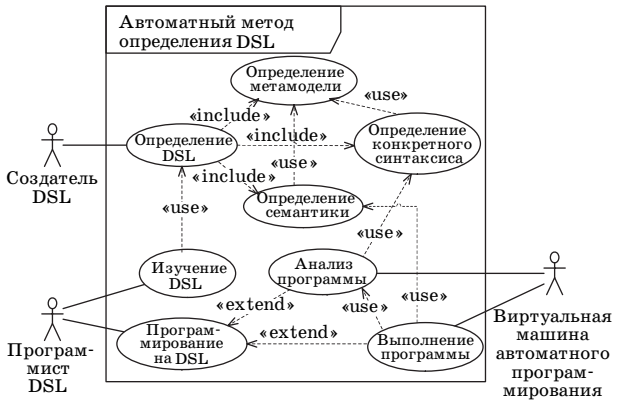
В автоматном методе мы предлагаем использовать следующую модель системы взаимодействующих автоматов (рис. 10, 11). Входным алфавитом автомата являются *события*, посылаемые *источником событий*. В зависимости от полученных событий и проверки *сторожевых условий* на переходах автомат меняет свое текущее состояние и выполняет *действия* на переходах и в состояниях (*эффекты*), адресованные *объекту управления*.

Мы придерживаемся принципа Бертрана Мейера [5] разделения операций интерфейса на *запросы*, доставляющие значения и не меняющие состояния объекта, и *команды*, меняющие состояние объекта. Кроме того, мы считаем обязательным указание для каждого объекта не только *предоставляемых*, но и *требуемых* интерфейсов. Таким образом, мы определяем все четыре возможных интерфейса взаимодействия между автоматом, его источником событий и объектом управления:

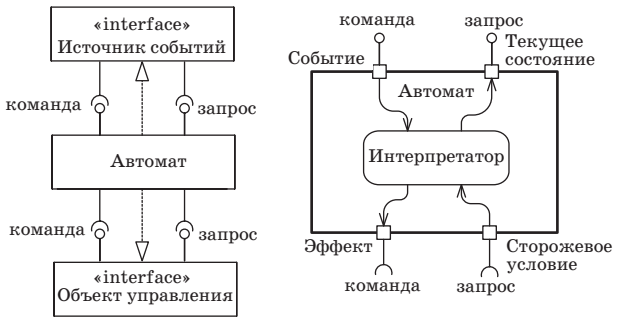
- события на переходах являются предоставляемыми командами автомата;
- сторожевые условия на переходах используют требуемые запросы к объекту управления;
- эффекты — это требуемые команды объекта управления;
- источник событий может использовать предоставляемые запросы о текущем состоянии автомата (см. рис. 10)[5].

Источником событий и объектом управления могут быть как внешние (по отношению к системе автоматов) объекты, так и этот же или другой автомат системы. Заметим, что унифицированная трактовка управляющих автоматов, объектов управления и источников событий является одной из наиболее существенных отличительных особенностей нашего метода по сравнению с существующими моделями автоматного программирования [4, 6].

Используемая нами структура (модель) автомата схожа с метамоделью UML [7] (см. рис. 11).



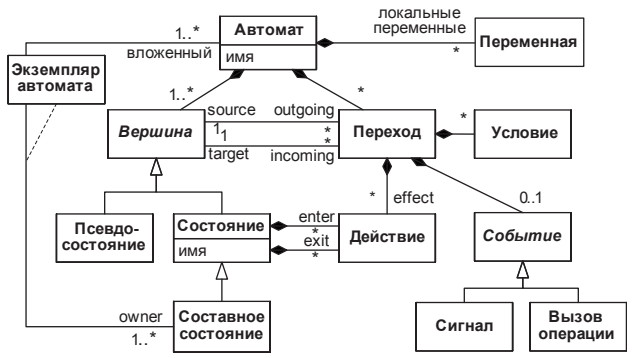
■ ***Рис. 9.*** *Модель использования автоматного метода*



■ ***Рис. 10.*** *Взаимодействие и внутреннее устройство автомата*

[4] Нумерация рисунков продолжает нумерацию первой части статьи.

[5] В наших примерах мы не используем эту возможность, но оставляем ее в модели системы взаимодействующих автоматов для общности.



■ ***Рис. 11.*** *Модель автомата*

Основной особенностью предлагаемой модели системы автоматов являются понятия *класса* и *экземпляра* автомата. Каждая диаграмма автомата UML задает класс автоматов. Конкретный автомат (экземпляр класса) создается явным образом и интерпретируется в процессе выполнения[6].

Взаимодействие между автоматами системы осуществляется с помощью экземпляров автоматов, вложенных в составные состояния [9]. При первом переходе в составное состояние создается новый экземпляр вложенного автомата. При последующих переходах используется уже созданный экземпляр. При этом допускается рекурсивное вложение автоматов, т. е. в составное состояние может быть вложен экземпляр автомата того же класса автоматов. Исходящие переходы составного состояния наследуются всеми состояниями вложенного автомата, но могут быть в нем переопределены.

Допускается задание в автоматах локальных переменных, которые могут использоваться в сторожевых условиях и в действиях на переходах автомата. Типом локальных переменных могут быть типы, определенные в метамодели языка, или встроенные типы UML.

Мы предполагаем, что система автоматов содержит единственный головной автомат системы, с которого начинается выполнение этой системы[7].

Определение конкретного синтаксиса системой распознающих автоматов

В автоматном методе определения проблемно-ориентированных языков не используется традиционное грамматическое описание. Мы описываем анализатор (распознаватель) языка в виде системы автоматов, общим входным алфавитом которых является множество представлений абстрактных знаков (терминалов языка, в терминологии формальных грамматик). С практической точки зрения мы считаем представлением абстрактного знака событие, посылаемое источником событий, используемым для создания и изменения программы. Например, в качестве элементов нотации языка могут выступать символы в тексте, геометрические фигуры на диаграмме, поля и кнопки диалоговых окон, ячейки электронной таблицы, звуки произнесенной команды и т. д. Соответственно источником событий может быть лексический анализатор текста, или графический редактор диаграмм, или редактор формул, или диалоговое окно — любой источник событий. Такой подход реализует описанную выше идею использования для представления программы различных нотаций, а также их комбинаций (см. ч. 1, разд. «Назначение и область применения метода»).

Наиболее характерным для автоматного метода является задание конкретного синтаксиса исходя из метамодели языка. Система конечных автоматов для распознавания языка строится по метамодели языка с помощью следующих эмпирических правил.

- Каждому классу с именем A в метамодели языка соответствует класс автоматов с именем A SM. Экземпляр автомата A SM в процессе выполнения строит экземпляр класса A, т. е. является его конструктором.
- Начальной структурной единице (аксиоме) языка соответствует головной автомат системы.
- Каждой составляющей B класса A в автомате A SM соответствует составное состояние, в которое вложен автомат B SM. Для наглядности имя этого составного состояния совпадает с именем роли класса B в композиции «B есть часть A» или с именем атрибута B в классе A[8].
- Альтернативной декомпозиции классов соответствует альтернативная декомпозиция автоматов, а именно: в каждое составное состояние вкладывается отдельный экземпляр автомата указанного класса.
- Дизъюнктивной композиции классов, перечислимым типам и обобщению классов соответствуют сегментированные переходы на диаграмме автомата.
- Кратности полюса и массивам соответствует петля на диаграмме автомата.

По умолчанию, объектом управления каждого автомата является конструируемый им экземпляр класса метамодели языка. Этот объект обо-

[6] Технические детали программной реализации автоматного метода (виртуальной машины автоматного программирования) рассмотрены в других наших статьях, опубликованных [8] и готовящихся к печати.

[7] Считается, что головной автомат вложен в анонимное состояние, принадлежащее машине автоматного программирования.

[8] Поэтому мы разрешаем прямо использовать имя атрибута B в автомате A SM.

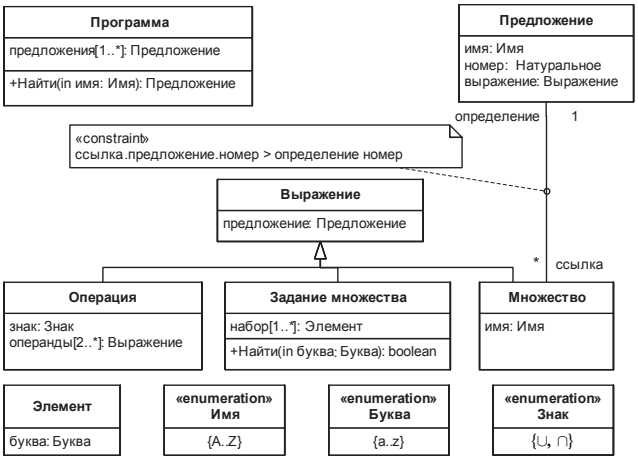
значается ключевым словом this, которое можно опускать в контексте данного автомата.

При таком задании конкретного синтаксиса метамодель целесообразно представить в виде набора классов с использованием атрибутов, а не в виде безымянных композиций, что облегчает конструирование автоматов.

Приведем пример задания текстового синтаксиса для мини-языка множеств. На рис. 12 представлена метамодель мини-языка множеств, пересмотренная с учетом изложенных рекомендаций.

Применяя изложенные выше правила к метамодели на рис. 12, получаем полную систему автоматов (рис. 13–15).

На рис. 13 использованы следующие элементы нотации. Обозначение предложения[k] подразумевает, что каждый элемент массива предложения разбирается отдельным экземпляром автомата класса Предложение SM. Напомним, что переход из составного состояния наследуется всеми состояниями вложенного автомата и может быть переопределен в нем. Знаком * обозначается любое событие. Таким образом, обрабатываются все неожиданные события, в том числе ошибки. Все события в этих автоматах — это терминалы разбираемого языка. Поэтому вместо сложного выражения вида «lex.getc(out token) [token == 'c']» мы пишем просто «c» на дуге перехода.



■ ***Рис. 12.*** *Пересмотренная метамодель мини-языка множеств*

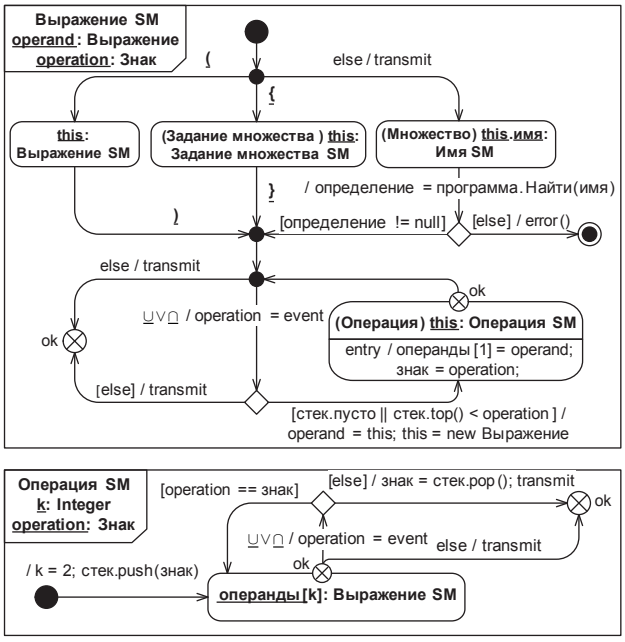
■ ***Рис. 13.*** *Головной и основной автоматы, распознающие мини-язык множеств*

На рис. 14 применяется еще ряд допустимых обозначений. Для ветвления по событиям используется переходное состояние (junction state [9]) и ключевое слово else, которое означает любое событие, кроме альтернативных. Ключевое слово transmit означает, что обрабатываемое на данном переходе событие снова становится текущим и должно быть обработано еще раз. Тем самым моделируется делегирование событий. Мы допускаем расширение обозначений UML, позволяя записывать объединение событий, например $\cup \vee \cap$; а также сохранение текущего события в локальной переменной, например operation = = event.

Для ветвления по сторожевому условию используются состояния выбора (decision state [9]). При этом предикат [else] имеет обычный смысл дополняющего условия.

Кроме того, в автомате Выражение SM экземпляр класса Выражение конструируется либо как Операция, либо как Задание множества, либо как Множество, т. е. происходит конкретизация типа экземпляра класса Выражение. Для этого используется обозначение вида (Операция) this, где Операция — подкласс класса Выражение.

Необходимо объяснить, как в этих автоматах учитывается приоритет операций, который относится именно к конкретному синтаксису. Заметим, что перечислимые типы считаются линейно упорядоченными. Тем самым определение пере-



■ ***Рис. 14.*** *Автоматы, реализующие разбор выражения с учетом приоритетов операций*

числимого типа Знак на рис. 12 позволяет удобно передать информацию о приоритетах операций. Для хранения последовательности знаков операций используется внешний объект Стек. Если операция первая или ее приоритет выше предыдущей, то создается новое выражение и построенное выражение запоминается в массиве операндов. Если же операция совпадает с предыдущей, то продолжается разбор в том же экземпляре выражения без выхода из автомата Операция SM. В противном случае, т. е. если приоритет текущей операции ниже предыдущей, происходит возврат из рекурсии на уровень выше (в автомат Выражение SM), где создается новое выражение и построенная операция запоминается в массиве операндов.

Заметим, что на рис. 15 в последнем автомате нет заключительного состояния, потому что завершение работы происходит по унаследованному от внешнего автомата переходу по событию }.

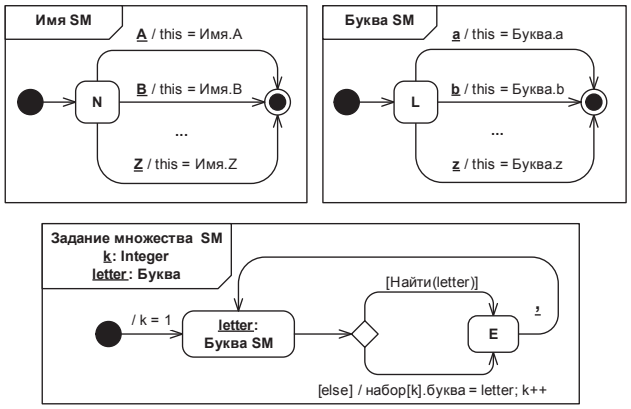
Объектами, внешними по отношению к описанной системе автоматов, являются Лексический анализатор (является источником событий) и Стек (является объектом управления и используется как рабочая память), представленные на рис. 16.

Рассмотрим теперь язык описания шахматных позиций. Пример диаграммы, описывающей шахматную позицию, приведен на рис. 17. Нетрудно представить себе, что имеется интерактивное приложение, в котором есть панель фигур и пешек, изображение доски, и пользователь перетаскивает фигуры и пешки с панели на доску, формируя тем самым позицию. Терминалами этого языка являются события перетаскивания конкретной фигуры на конкретное поле доски.

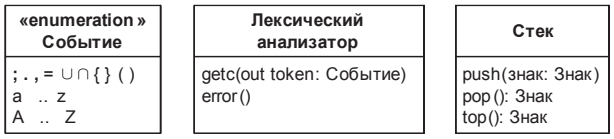
Таким образом, источником событий является графический интерфейс пользователя, спецификация которого показана на рис. 18. Здесь мы разделяем панель фигур и пешек на две панели — для черных и для белых, и каждая из них является экземпляром абстрактного класса *Chess diagram constructor*. События выбора конкретной фигуры (например, щелчком кнопки мыши) мы обозначаем изображением этой фигуры. Событие опускания фигуры на конкретное поле (например, щелчком кнопки мыши) обозначается значком ☐. При этом действует правило «взялся — ходи», т. е. эти действия всегда выполняются парой. Событие завершения ввода позиции обозначено значком ☒.

В метамодели языка описания шахматных позиций (см. ч. 1, рис. 4 и 6) мы устраняем безымянные композиции и ненужные ассоциации, получая метамодель, представленную на рис. 19.

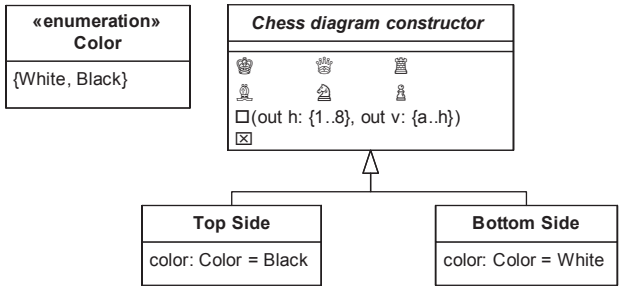
■ ***Рис. 17.*** *Пример диаграммы шахматной позиции*



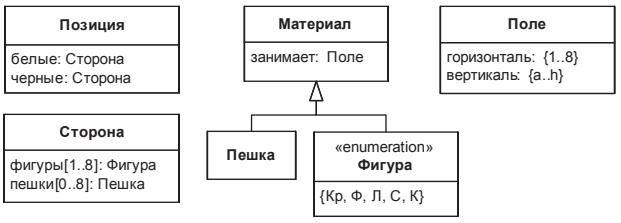
■ ***Рис. 15.*** *Автоматы, реализующие разбор элементарных конструкций*



■ ***Рис. 16.*** *Лексический анализатор и стек*



■ ***Рис. 18.*** *Источник событий для анализатора языка описания шахматных позиций*

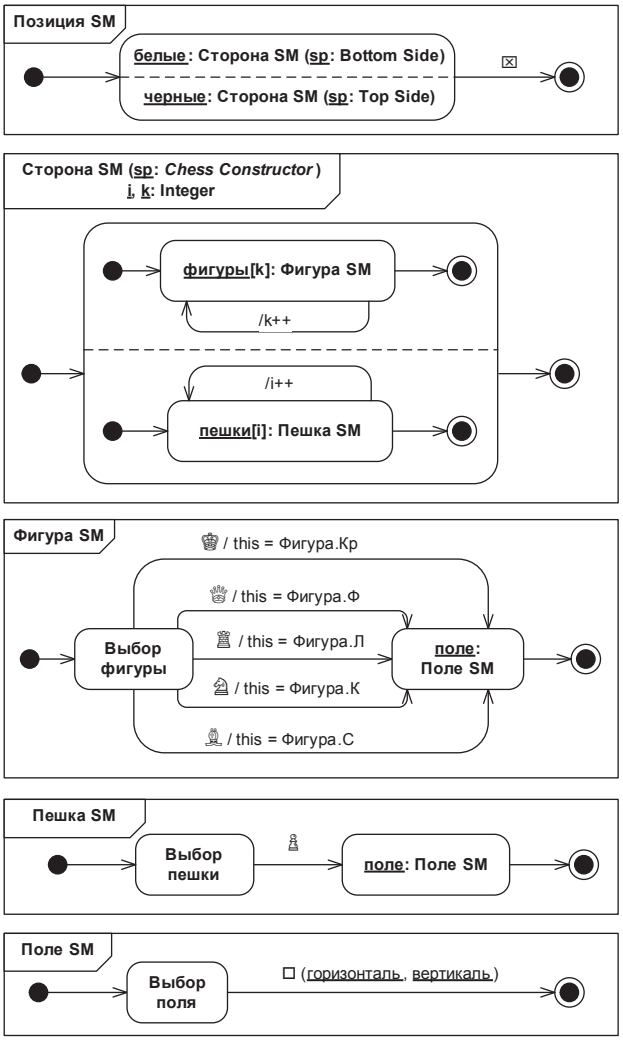


■ ***Рис. 19.*** *Пересмотренная метамодель языка описания шахматных позиций*

Далее, руководствуясь правилами преобразования классов метамодели в автоматы, изложенными в начале раздела, получаем систему автоматов (рис. 20). Важно, что фигуры и пешки можно расставлять в произвольном порядке, что передается с помощью ортогонального составного состояния [9] (автоматы Позиция SM и Сторона SM).

В предыдущих двух примерах систему автоматов построили исходя из только метамодели языка, без привлечения иных формализмов. Однако мы вовсе не противопоставляем автоматный метод и использование традиционных формализмов при задании конкретного синтаксиса. Если синтаксис языка *уже* задан, например с помощью порождающей грамматики или синтаксических диаграмм Вирта [10], то автоматный метод позволяет эффективно использовать эту информацию.

Рассмотрим набор синтаксических диаграмм Вирта (рис. 21), описывающих тот же мини-язык

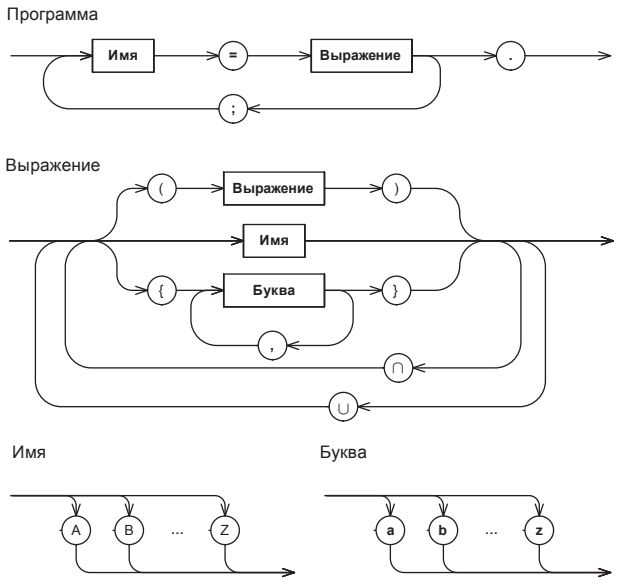


■ ***Рис. 20.*** *Система автоматов, позволяющих задать диаграмму шахматной позиции*

множеств, что и грамматика в разделе «Определение отношений метамодели языка» (см. ч. 1). Эти диаграммы получены почти автоматически из грамматики: сначала для каждого правила грамматики были нарисованы очевидные диаграммы, а потом однократно используемые нетерминалы подставлены в соответствующие диаграммы.

Система автоматов, распознающих конкретный синтаксис, получается автоматически из системы синтаксических диаграмм путем выполнения следующих преобразований:

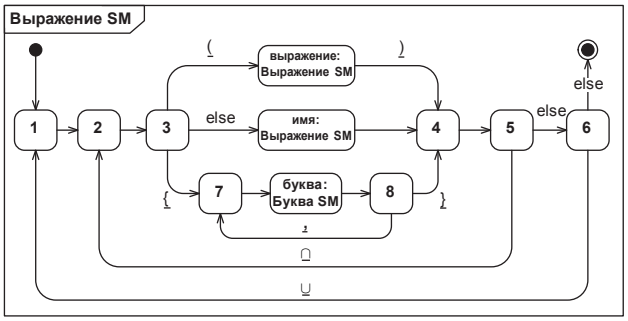
- каждая синтаксическая диаграмма преобразуется в диаграмму автомата;
- вводятся начальное состояние, к которому присоединяется входная стрелка, и заключительное состояние, к которому присоединяется выходная стрелка;
- нетерминалы преобразуются в составные состояния, в которые вложены автоматы соответствующих конструкций;
- в каждой точке разветвления или слияния дуг синтаксической диаграммы вводится служебное состояние[9];
- терминалы переносятся на дуги перехода в качестве событий;
- немотивированные переходы, альтернативные мотивированным, помечаются ключевым словом else[10].



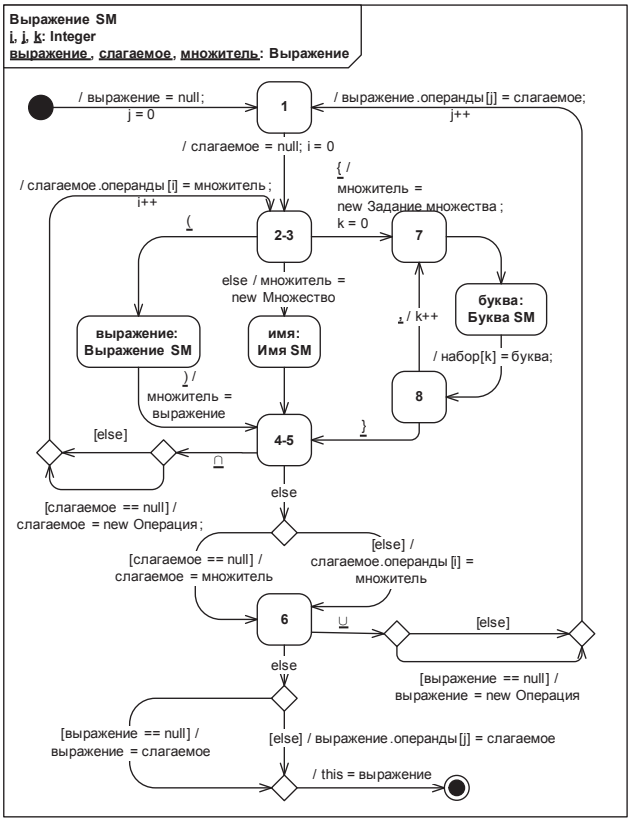
■ ***Рис. 21.*** *Синтаксические диаграммы мини-языка множеств*

[9] Некоторые служебные состояния могут оказаться излишними, их следует удалить после того, как будут построены действия на переходах.

[10] Предполагается, что для каждого состояния имеется не более одного немотивированного исходящего перехода, т. е. что автомат детерминированный.



■ ***Рис. 22.*** *Автомат-распознаватель конструкции Выражение*



■ ***Рис. 23.*** *Автомат-построитель конструкции* Выражение

Пример результата такого преобразования для самого сложного нетерминала Выражение приведен на рис. 22. Этот автомат является распознавателем конструкции Выражение, т. е. он переходит в заключительное состояние тогда и только тогда, когда на вход ему подается корректное выражение мини-языка множеств.

Автомат-конструктор класса Выражение показан на рис. 23. Он получен из автомата-распознавателя добавлением действий на переходах, причем некоторые действия являются условными, поэтому применены сегментированные переходы. Кроме того, объединены эквивалентные состояния *2* и *3*, *4* и *5*.

Мы видим, что наличие традиционного формального описания синтаксиса резко упрощает построение системы автоматов распознавания.

Вместе с тем система автоматов разбора ничуть не становится проще по причине того, что структура системы автоматов никак не связана в данном случае со структурой конструируемого объекта, и ее приходится учитывать вручную при программировании действий на переходах[11]. Это еще одно свидетельство в пользу целесообразности рассматривать абстрактный синтаксис как главное и первичное описание языка.

Окончание следует

Литература

1. **Ершов А. П.** Предисловие редактора перевода // Пересмотренное сообщение об Алголе-68. — М.: Мир, 1979. — С. 5–8.
2. **Непейвода Н. Н.** Семантика алгоритмических языков // Итоги науки и техники. Сер. Теория вероятностей. Математическая статистика. Теоретическая кибернетика / ВИНИТИ. 1983. Т. 20. С. 95–166.
3. **Лавров С. С.** Программирование. Математические основы, средства, теория. — СПб.: БХВ-Петербург, 2001. — 317 с.
4. **Поликарпова Н. И., Шалыто А. А.** Автоматное программирование. — СПб.: Питер, 2008. — 176 с.
5. **Мейер Б.** Объектно-ориентированное конструирование программных систем. — М.: ИНТУИТ.ру; Русская Редакция, 2005. — 1232 с.
6. **Шалыто А. А.** SWITCH-технология. Алгоритмизация и программирование задач логического управления. — СПб.: Наука, 1998. — 628 с.
7. **OMG Unified Modeling Language** (OMG UML), Superstructure, Version 2.2. http://www.uml.org/ (дата обращения: 12.02.2010).
8. **Новиков Ф. А., Тихонова У. Н.** Определение проблемно-ориентированных языков интерпретируемыми автоматами // Научно-технические ведомости СПбГПУ. 2008. № 5(65). С. 93–98.
9. **Буч Г., Якобсон А., Рамбо Д.** UML. 2-е изд. Сер. Классика CS. — СПб.: Питер, 2005. — 736 с.
10. **Вирт Н., Йенсен К.** Паскаль: руководство для пользователя. — М.: Финансы и статистика, 1989. — 254 с.

[11] Впрочем, точно такой же недостаток присущ и традиционным компиляторам компиляторов.