УДК 681.3.06

АВТОМАТИЧЕСКОЕ ЗАВЕРШЕНИЕ ВВОДА УСЛОВИЙ В ДИАГРАММАХ СОСТОЯНИЙ

В. С. Гуров,
ведущий разработчик
М. А. Мазин,
ведущий разработчик
Компания ООО «Интелли Джей Лабс»
А. А. Шалыто,
доктор техн. наук, профессор
Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Описан процесс разработки системы автоматического завершения ввода условий на переходах в диаграммах состояний для программного пакета с открытым кодом UniMod. Основой этой системы является автомат Мили, который строится по грамматике, описывающей язык, используемый при задании условий на переходах.

Введение

В работе [1] предложен метод проектирования событийных объектно-ориентированных программ с явным выделением состояний, названный «SWITCH-технология» или «автоматно-ориентированное программирование». Особенность этого подхода состоит в том, что поведение таких программ описывается с помощью графов переходов структурных конечных автоматов с нотацией, предложенной в работе [2]. SWITCH-технология для описания каждого автомата определяет два типа диаграмм (схема связей и граф переходов). При наличии нескольких автоматов также строится схема взаимодействия автоматов. SWITCH-технология задает нотацию и операционную семантику используемых диаграмм.

Программный пакет с открытым кодом UniMod (http://unimod.sf.net), созданный авторами статьи [3], обеспечивает разработку и выполнение автоматно-ориентированных программ. Этот пакет позволяет использовать UML-нотацию при построении диаграмм в рамках SWITCH-технологии. При этом схемы связей, определяющие интерфейс автоматов, строятся в нотации диаграмм классов языка UML, а графы переходов — в UML-нотации диаграмм состояний. В состав пакета UniMod входит встраиваемый модуль (plug-in) для платформы Eclipse (http://www.eclipse.org), позволяющий создавать и редактировать UML-диаграммы классов и состояний, которые соответствуют схеме связей и графу переходов.

Как отмечено в работе [3], интегрированные системы для разработки программ предоставляют удобные средства для работы с кодом, такие как, например:

- подсветка семантических и синтаксических ошибок;
- автоматическое завершение ввода и автоматическое исправление ошибок;
- форматирование и рефакторинг кода [4];
- запуск и отладка программы внутри среды разработки.

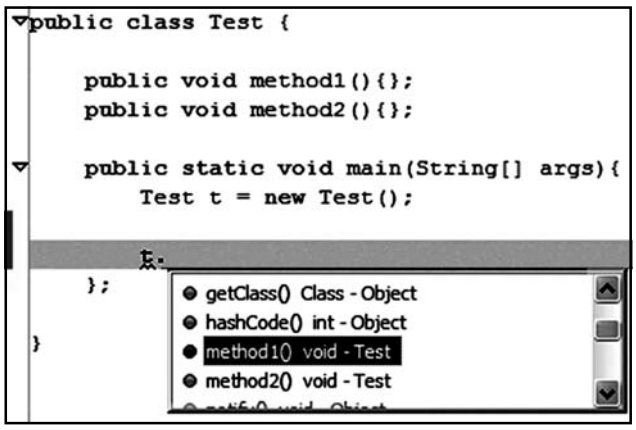
В английском языке эти средства получили название «code assist».

В рамках создания очередной версии пакета UniMod перед авторами встала задача реализации системы автоматического завершения ввода условий на переходах в диаграммах состояний. В статье описан процесс создания такой системы и ее автоматно-ориентированная реализация, выполненная с помощью предыдущей версии пакета UniMod.

Постановка задачи

Автоматическим завершением ввода, применительно к редактированию программных текстов, традиционно называют возможность, позволяющую пользователю получить список строк, при добавлении которых в текст после позиции курсора программа будет синтаксически верной.

Например, на рис. 1 показано, что среда разработки Eclipse при нажатии сочетания клавиш Ctrl-



■ ***Рис. 1.*** *Пример автоматического завершения ввода*

Space после строки текста t. предлагает на выбор варианты автоматического завершения ввода.

Сформулируем требования к проектируемой системе автоматического завершения ввода. Пусть задан язык L и на вход системы подана строка α.

1. Если строка α является префиксом некоторого предложения языка L ($\exists\omega$: $\alpha\omega \in L$), то система должна возвращать множество строк

$$C(\alpha) = \{\beta_i\}_{i=1..n},$$

любая из которых может являться продолжением строки α. Изложенное может быть записано соотношением вида

$$\forall i \in [1..n] \exists\gamma : \alpha\beta_i\gamma \in L.$$

2. Если строка α не является префиксом предложения на заданном языке ($\forall\omega : \alpha\omega \notin L$), то система должна с помощью дополнения строки недостающими символами или удаления лишних символов трансформировать строку в правильный префикс предложения языка. Количество дополнений и удалений должно быть как можно меньше.

Элементы теории построения компиляторов, применяемые в настоящей работе

Если исходный язык L задан порождающей грамматикой, то для построения такой системы необходимо использовать методы проектирования компиляторов [5]. Существует множество инструментов для автоматического создания компиляторов по заданной грамматике (http://www.kulichki.net/kit/tools/java.html). На рис. 2 приведена обобщенная структура компилятора.

Лексический анализатор осуществляет чтение входной цепочки символов и их группировку в элементарные конструкции, называемые лексемами.

Синтаксический анализатор выполняет разбор исходной программы, используя поступающие лексемы; семантический анализ программы; построение ее промежуточного представления.

Генератор кода преобразует это представление в объектный код. Генератор может быть заменен интерпретатором. При этом вместо создания объектного кода будет выполняться интерпретация промежуточного представления программы.

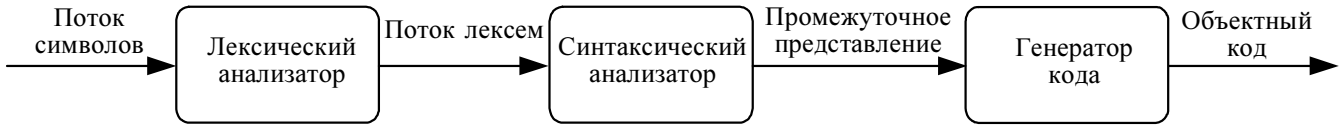
В проекте UniMod условия на переходах являются логическими формулами, в которых могут использоваться предикаты первого порядка определенного вида. Трансляция этих условий выполняется с помощью так называемого «компилятора компиляторов» ANTRL [6]. Он по заданной $LL(k)$-грамматике строит код на языке Java, реализующий лексический анализатор и рекурсивный нисходящий синтаксический анализатор [5]. Построенный синтаксический анализатор может быть использован и как распознаватель принадлежности выражения заданному грамматикой языку, и как транслятор выражений в абстрактное синтаксическое дерево. Данный анализатор не может быть применен для построения системы автоматического завершения ввода, так как в случае подачи ему на вход префикса для выражения на заданном языке вместо законченного выражения он выдает ошибку.

Существуют различные подходы к реализации требуемой системы. Одним из таких подходов является использование управляемого таблицей разбора нерекурсивного нисходящего синтаксического анализатора, построенного на основе автомата с магазинной памятью [5]. Граф переходов такого автомата имеет одну вершину и много петель.

Таблица разбора представляет собой двумерный массив $M[A, a]$, где A — нетерминал, a — терминал (лексема) или символ конца потока \$. В ячейках таблицы записываются продукции грамматики, с помощью которых заменяются нетерминалы на вершине стека. Пустые ячейки таблицы означают ошибки.

При подаче на вход этому анализатору строки α без символа конца потока анализатор остановится, имея какой-то нетерминал на вершине стека. В этом случае множество терминалов $C(\alpha)$, ожидаемых после обработанной строки, может быть определено как $\{\beta: M[T, \beta] \neq \varnothing\}$, где T — нетерминал на вершине стека после остановки анализатора.

Для реализации восстановления после ошибок в «режиме паники» [5] таблица разбора может быть дополнена синхронизирующими символами, которые вписываются в некоторые пустые ячей-



■ ***Рис. 2.*** *Обобщенная структура компилятора*

ки. При получении неожидаемого терминала анализатор пропускает символы входного потока до тех пор, пока не будет обнаружен терминал, соответствующий синхронизирующему символу. Для восстановления на уровне фразы (несколько взаимосвязанных лексем) в некоторые пустые ячейки вписываются указатели на подпрограммы обработки ошибок, которые могут изменять, вставлять или удалять терминалы входного потока или элементы стека.

В работе [7] показано, как создать программу нерекурсивного нисходящего синтаксического анализатора, используя автоматный подход. При этом таблица разбора применяется в роли объекта управления.

Цель настоящей работы состоит в создании системы автоматического завершения ввода, позволяющей исключить таблицу разбора нисходящего нерекурсивного синтаксического анализатора, которую строить трудно.

Описывается также разработанный алгоритм восстановления разбора строки после ошибок на уровне фразы.

Описание предлагаемого подхода

Предлагаемый подход состоит в том, что для заданной контекстно-свободной грамматики типа *LL*(*1*) строится конечный автомат Мили, который будет являться синтаксическим анализатором.

В отличие от классического подхода предлагается этот автомат строить как реагирующий на события, что объясняется тем, что в дальнейшем автомат будет реализован диаграммой состояний языка UML, которая по спецификации является событийной [8]. Поэтому автомат будет реагировать на события, которые поставляет ему лексический анализатор. Каждому событию соответствует терминал.

В работах [5, 9] нисходящий синтаксический анализатор также строится на основе диаграмм состояний, однако, в отличие от предлагаемого подхода:

- для каждого правила вывода, входящего в описание грамматики, строится одна диаграмма;
- в таких диаграммах отсутствуют события и выходные воздействия. Отсутствие выходных воздействий объясняется наличием срединной рекурсии, которая появляется на диаграммах, строящихся для правил вывода вида $S \to aSb$;
- при наличии срединной рекурсии в правилах вывода на диаграммах существуют дуги, помеченные нетерминалами, что не позволяет полностью заменить описание языка с грамматики на автомат;
- в указанных выше работах реализация диаграмм не описывается.

Описываемый в настоящей статье подход предлагает сворачивать все диаграммы в одну, при необходимости удаляя срединную рекурсию с помощью метода, описанного в работе [10]. Это дает возможность избавиться от упоминания нетерминалов на диаграммах состояний и, следовательно, разорвать семантическую связь с исходной грамматикой. Такой разрыв позволит описывать язык только с помощью диаграммы состояний и автоматически получать реализацию распознавателя для данного языка.

При подаче на вход системе, построенной описанным выше образом, незавершенной строки автомат, реализующий синтаксический анализатор, останавливается в каком-то состоянии. События, заданные на переходах из состояния, в котором остановился автомат, определяют множество терминалов, которые могут следовать за последним терминалом, извлеченным из входной строки.

После построения такого множества терминалов каждый терминал этого множества обратно преобразуется лексическим анализатором в строку символов для отображения пользователю.

Построение диаграммы состояний синтаксического анализатора

Пусть *LL*(*1*)-грамматика для нашего примера задана следующим множеством правил вывода:

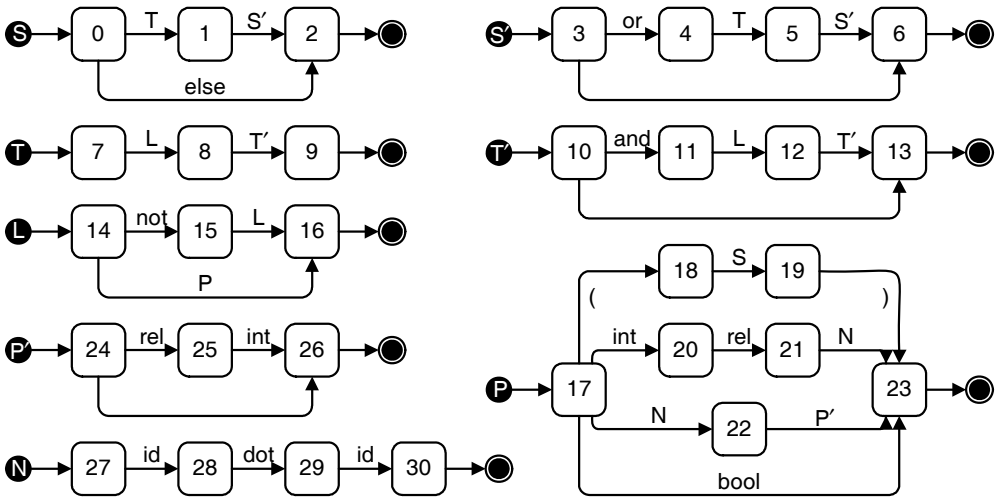
1. $S \to$ **else** $\mid T\ S'$
2. $S' \to$ **or** $T\ S' \mid \varepsilon$
3. $T \to L\ T'$
4. $T' \to$ **and** $L\ T' \mid \varepsilon$
5. $L \to$ **not** $L \mid P$
6. $P \to$ **'('** S **')'** $\mid$ **int rel** $N \mid$ **bool** $\mid N\ P'$
7. $P' \to$ **rel int** $\mid \varepsilon$
8. $N \to$ **id dot id**

Терминал **id** соответствует идентификатору, **int** — целочисленной константе, **bool** — булевской константе, **rel** — бинарному отношению (‘>’, ‘<’, ‘>=’, ‘<=’, ‘=’, ‘≠’), терминалы **and**, **or**, **not** — булевским операциям, терминал **else** — оператор «иначе». Опишем формальный процесс построения автомата Мили для данной грамматики.

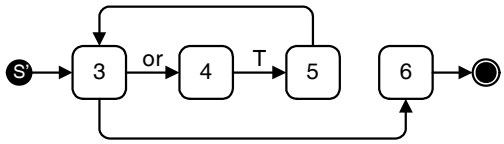
На рис. 3 для каждого нетерминала заданной грамматики показаны диаграммы состояний, построенные с помощью метода, описанного в работе [5]. Приведенные диаграммы, со сквозной нумерацией состояний, отличаются от аналогичных диаграмм, используемых в работе [5], наличием начального и конечного состояний. Из начального состояния всегда существует только один переход. В конечное состояние также всегда ведет один переход.

Состояния в приведенных диаграммах соответствуют позициям [11] в правилах вывода, метки на переходах — терминалам и нетерминалам, отделяющим позиции друг от друга. Если нетерминал выводит ε-правило (пустое), то из состояния, соответствующего начальной позиции, существует непомеченный переход в состояние, соответствующее конечной позиции. Непомеченные переходы называются также немотивированными.

Далее множество диаграмм, представленных на рис. 3, будем преобразовывать в одну диаграмму состояний, на которой все переходы помечают-



■ ***Рис. 3.*** *Диаграммы состояний для каждого нетерминала грамматики*



■ ***Рис. 4.*** *Удаление правой рекурсии на диаграмме состояний для нетерминала S′*

ся только терминалами. Процесс такого преобразования предполагает выполнение следующих шагов:

- удаление правой рекурсии;
- удаление немотивированных переходов;
- подстановка диаграмм состояний друг в друга;
- удаление срединной рекурсии.

Опишем каждый из этих шагов.

Удаление правой рекурсии

Наличие праворекурсивного правила вывода означает, что на диаграмме, соответствующей некоторому терминалу N, существует переход, помеченный тем же нетерминалом N, который ведет в состояние, соответствующее конечной позиции.

Например, правая рекурсия в правиле 2 приводит к наличию перехода из состояния *5* в состояние *6* на соответствующей диаграмме состояний (см. рис. 3). Для устранения этой рекурсии указанный переход заменим немотивированным переходом в состояние, соответствующее начальной позиции, — в состояние *3* (рис. 4).

Удаление немотивированных переходов

Наличие немотивированного перехода из состояния S_i в состояние S_j означает, что за позицией, соответствующей состоянию S_i, могут следовать те же терминалы и нетерминалы, что и за позицией, соответствующей состоянию S_j.

Для устранения немотивированного перехода выполняются следующие операции:

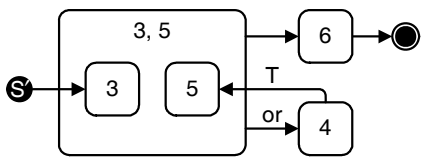
- создается составное состояние $S_{i,j}$, исходящие дуги которого заменяют одинаковые исходящие дуги для всех вложенных состояний [8];
- состояния S_i и S_j помещаются внутрь состояния $S_{i,j}$;
- все переходы из состояния S_i заменяются аналогичными переходами из состояния $S_{i,j}$.

На рис. 5 показано удаление немотивированного перехода из состояния *5* в состояние *3* (см. рис. 4).

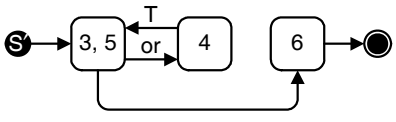
Диаграмма на рис. 5 может быть упрощена за счет ликвидации эквивалентных состояний *3* и *5*, имеющих одинаковые исходящие переходы (рис. 6).

Отметим, что это преобразование аналогично вычеркиванию одинаковых записей из таблицы, задающей функцию переходов автомата при минимизации числа состояний в нем [12].

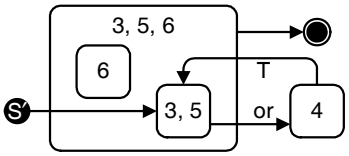
Теперь удалим немотивированный переход в полученной диаграмме (рис. 7).



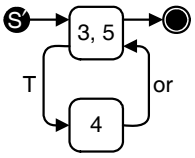
■ ***Рис. 5.*** *Удаление немотивированного перехода из состояния 5 в состояние 3*



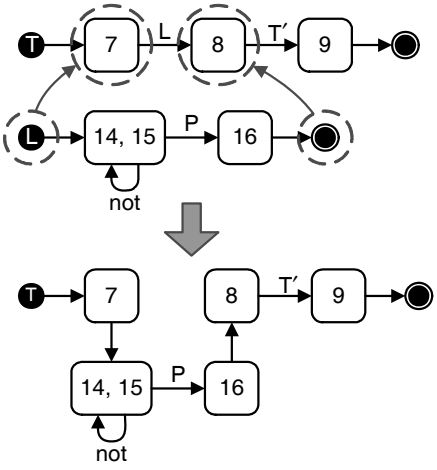
■ ***Рис. 6.*** *Упрощение диаграммы состояний, показанной на рис. 5*



■ ***Рис. 7.*** *Удаление немотивированного перехода из состояния 3, 5 в состояние 6*



■ ***Рис. 8.*** *Преобразованная диаграмма состояний для нетерминала S′*



■ ***Рис. 9.*** *Подстановка диаграмм состояний друг в друга*

Состояние *6* (см. рис. 7) не имеет входящих переходов и, следовательно, не достижимо. Поэтому может быть удалено. После его удаления в состоянии *3, 5, 6* остается одно вложенное состояние *3, 5*. Начала переходов, исходящих из состояния *3, 5, 6*, следует перенести в состояние *3, 5*, а само состояние *3, 5, 6* удалить, так как оно теперь не имеет исходящих переходов. В результате получается диаграмма состояний для нетерминала S' (рис. 8).

Подстановка диаграмм состояний друг в друга

Количество диаграмм состояний может быть сокращено путем подстановки одних диаграмм в другие, что может привести, в том числе, и к одной диаграмме.

Предположим, что на диаграмме для нетерминала N_i существует переход из состояния S_i в состояние S_j, помеченный нетерминалом N_j. Заменим такой переход на немотивированный из состояния S_i в состояние, следующее за начальным на диаграмме состояний для нетерминала N_j. Добавим переход из состояния, предшествующего конечному, на диаграмме состояний для нетерминала N_j в состояние S_j. Отметим, что указанную подстановку требуется выполнять, только если эти нетерминалы разные ($N_i \neq N_j$). Это объясняется тем, что в противном случае имеет место срединная рекурсия, удаление которой будет описано ниже.

После выполнения такой подстановки возникшие немотивированные переходы следует устранить, как описано выше. При этом сначала устраняется немотивированный переход из состояния S_i, а затем немотивированный переход в состояние S_j.

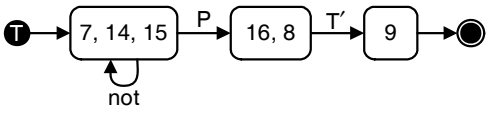
Рис. 9 иллюстрирует подстановку диаграммы состояний для нетерминала L в диаграмму состояний для нетерминала T.

На рис. 10 показана диаграмма состояний после устранения немотивированных переходов.

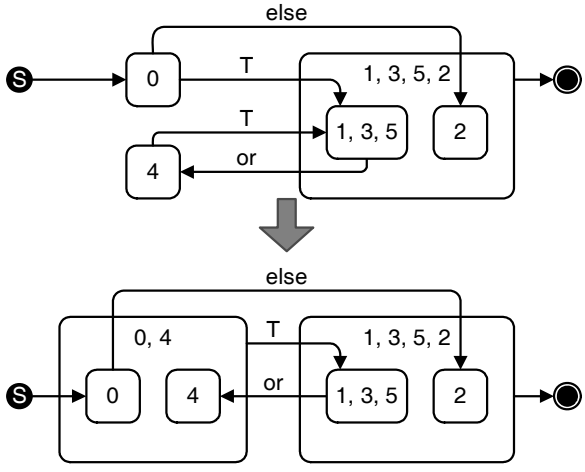
Выше был рассмотрен случай, когда подставляемый нетерминал встречался один раз.

Если некоторый нетерминал N_i присутствует на диаграмме для нетерминала N_j более одного раза, то каждый переход, помеченный символом N_i, необходимо заменить соответствующей диаграммой состояний. В результате количество однотипных подграфов на диаграмме N_j чрезмерно возрастает.

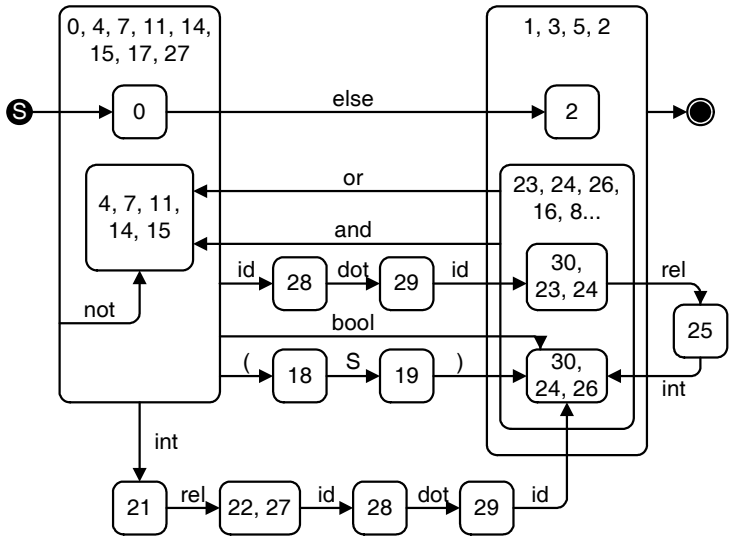
Предлагается преобразовать диаграмму N_j таким образом, чтобы нетерминал N_i встречался на ней минимальное число раз. Для этого использу-



■ ***Рис. 10.*** *Устранение немотивированных переходов после подстановки*



■ ***Рис. 11.*** *Удаление однотипных переходов*



■ ***Рис. 12.*** *Диаграмма состояний для рассматриваемой грамматики*

ется следующий подход: если несколько переходов, помеченных нетерминалом N_i, ведут в одно и то же состояние S, то можно выделить составное состояние и заменить все эти переходы единственным переходом, помеченным нетерминалом N_i, который исходит из группового состояния и входит в состояние S.

На рис. 11 данный подход применен для диаграммы состояний нетерминала S. Из рисунка следует, что два перехода, помеченных нетерминалом T, преобразованы в один переход.

Для описанной выше грамматики исходное множество диаграмм (см. рис. 3) преобразуется в одну диаграмму, приведенную на рис. 12.

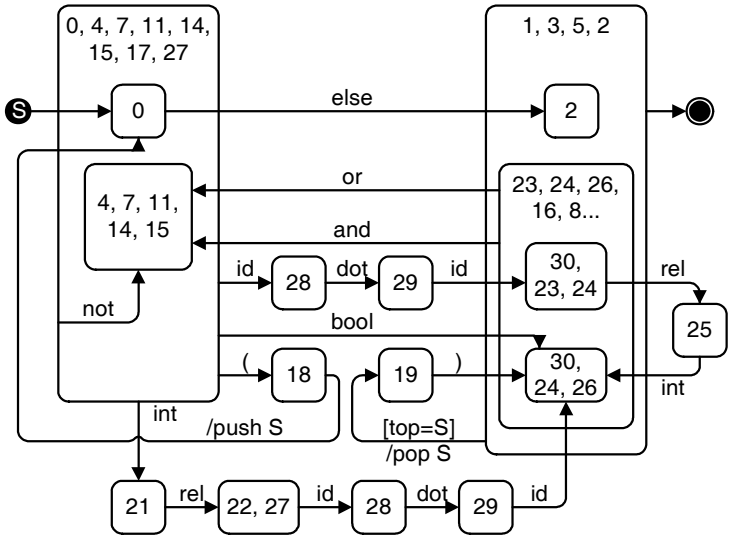
Поясним внутренний переход в крайнем левом составном состоянии. Он заменяет два перехода — из верхнего состояния в нижнее и петлю в нижнем состоянии.

В заключение отметим, что на полученной диаграмме присутствует срединная рекурсия — переход из состояния *18* в состояние *19*, так как дуга между этими состояниями помечена нетерминалом S — нетерминалом, для которого построена эта диаграмма в целом.

Удаление срединной рекурсии

Для удаления срединной рекурсии предлагается использовать подход, предложенный в работе [10], который основан на явном применении стека:

• пусть на диаграмме состояний для нетерминала N существует рекурсивный переход из состояний S_i в состояние S_j, помеченный нетерминалом N;



■ ***Рис. 13.*** *Диаграмма состояний с удаленной срединной рекурсией*

• заменим такой переход двумя немотивированными переходами. При этом первый из них должен выходить из состояния S_i, а входить в состояние, следующее за начальным состоянием на диаграмме. Второй переход должен выходить из состояния, предшествующего конечному, а входить в состояние S_j;

• на первом переходе должно выполняться действие по добавлению в стек метки M_{Sj}, соответствующей исходному целевому состоянию S_j, а на втором переходе — действие по извлечению метки M_{Sj} из стека при условии, что эта метка находится на вершине стека.

Таким образом, на диаграмме состояний появятся действия, которые воздействуют на стек.

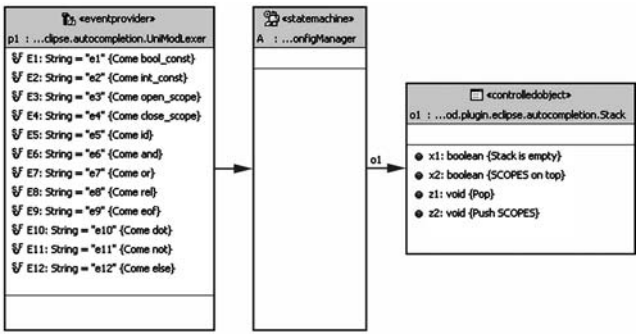
На рис. 13 показана диаграмма, приведенная на рис. 12, с удаленной срединной рекурсией и появившимися действиями, выполняемыми на переходах (автомат Мили).

Таким образом, заданную грамматику удалось заменить одной диаграммой состояний.

Использование построенной диаграммы

Будем строить систему автоматического завершения ввода с помощью инструментального средства UniMod [3, 13, 14]. Отметим, что здесь имеет место «рекурсия», так как рассматриваемая система для инструментального средства UniMod строится с помощью этого же средства.

При этом для построения системы автоматического завершения ввода сначала требуется создать схему связей автомата в виде UML-диаграммы классов. Эта схема строится следующим образом. В качестве источника событий используется лексический анализатор, события *e* которого соответствуют терминалам, которыми помечены переходы в построенной диаграмме состояний. В качестве объекта управления используется стек, который реализует входные *x* и выходные *z* воздействия, используемые в качестве действий и условий на построенной диаграмме состояний.

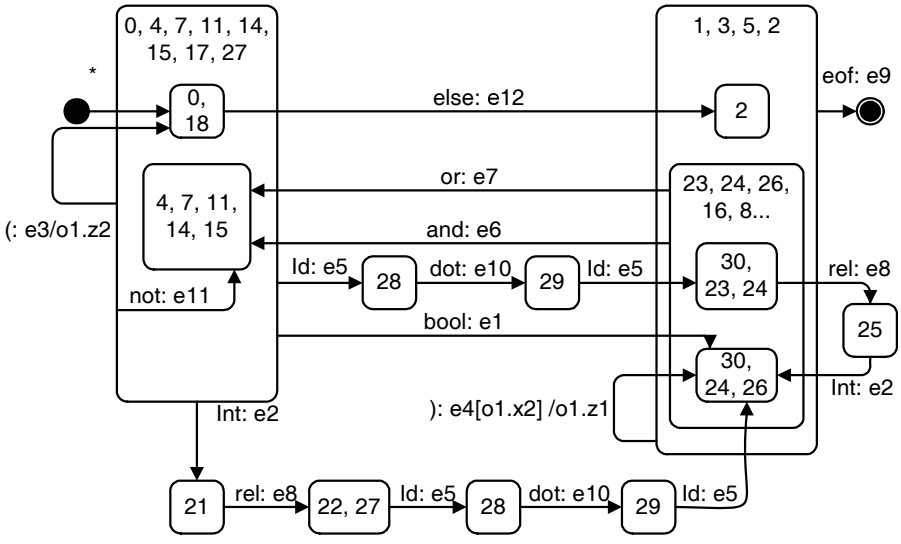


■ ***Рис. 14.*** *Схема связей автомата*

На рис. 14 приведена схема связей для рассматриваемого примера. В ней слева показан лексический анализатор, в центре — автомат, а справа — стек. При программировании источник событий и объект управления реализуются каждый своим классом. При этом в соответствии с работой [3] код для автомата может и не строиться, так как автомат может интерпретироваться.

На рис. 15 показана UML-диаграмма состояний автомата, построенная на основе диаграммы, приведенной на рис. 13, с помощью замены терминалов событиями и замены действий на переходах ссылками на методы объекта управления *o1*. Состояния *18* и *19* на рис. 15 отсутствуют из-за удаления немотивированных переходов.

Полученная модель системы состоит из двух UML-диаграмм (см. рис. 14, 15) и описывает распознаватель для языка, заданного приведенной выше грамматикой. Отметим, что информация о приоритете операций была потеряна в ходе



■ ***Рис. 15.*** *Диаграмма состояний автомата*

преобразований, поэтому модель может быть использована для распознавания принадлежности выражений языку, но не для трансляции выражений.

Выражение, принадлежащее языку, поданное на вход распознавателю, приводит автомат A в финальное состояние. При подаче на вход выражения, являющегося префиксом какого-либо выражения, принадлежащего языку, автомат остановится в каком-то состоянии, множество исходящих переходов из которого определяет множество возможных следующих терминалов.

Если выражение языку не принадлежит и не является префиксом какого-либо выражения, принадлежащего языку, то автомат A остановится в состоянии, в котором было получено событие и для которого не существовало исходящих переходов при текущих значениях входных переменных. В этом случае множество возможных следующих терминалов можно определить только для последнего правильно обработанного терминала.

Восстановление после ошибок

Перейдем к реализации второго требования, предъявляемого к системе, — обработке ошибочных строк. Для этого автомат A необходимо модифицировать таким образом, чтобы он корректно восстанавливался в случае подачи на вход выражения, не являющегося префиксом какого-либо выражения, принадлежащего языку. При этом автомат должен всегда останавливаться в состоянии, в которое существует переход по событию, соответствующему последнему терминалу, извлеченному из поданного на вход выражения.

Существует несколько возможных вариантов реализации восстановления автомата после ошибки. Например, можно для каждого состояния добавить такой исходящий переход, ведущий в конечное «ошибочное» состояние, что он будет срабатывать в случае отсутствия какого-либо другого исходящего перехода для пришедшего события и текущих значений входных переменных. Это приведет к тому, что при появлении в процессе распознавания первого же ошибочного терминала автомат завершит работу в «ошибочном» состоянии. Однако из «ошибочного» состояния нет исходящих переходов, и, следовательно, множество возможных последующих терминалов будет пустым.

Авторы предлагают использовать альтернативный алгоритм обработки ошибочных ситуаций, основанный на локально оптимальной коррекции входного потока терминалов от лексического анализатора. Такой подход называют также восстановлением на уровне фразы [5].

Пусть из состояния S нет исходящего перехода для пришедшего события e, соответствующего некоторому терминалу. Тогда коррекция потока может осуществляться двумя способами: дополнением потока недостающими терминалами; пропуском лишних терминалов в потоке.

Для того чтобы автомат A, находясь в состоянии S, пропустил в потоке терминал, соответствующий пришедшему событию e, необходимо добавить в автомат петлю в состоянии S по событию e. Тогда, находясь в состоянии S и получив событие e, автомат останется в состоянии S — проигнорирует пришедшее событие и как следствие пропустит терминал в потоке.

Для того чтобы автомат A, находясь в состоянии S, дополнил поток недостающими терминалами, следует выполнить указанные ниже операции:

1) найти достижимое из S состояние S_h такое, что в нем существует исходящий переход по событию e. Если из состояния S достижимы несколько таких состояний, то выберем ближайшее из них;

2) если из ближайшего найденного состояния S_h есть переход в некоторое состояние S_t по событию e при условии c, то необходимо добавить в автомат переход из состояния S в состояние S_t по событию e при условии c. Отметим, что отсутствие условия трактуется как тождественная истина.

Последовательность терминалов, соответствующих событиям, которыми помечен кратчайший путь из состояния S в состояние S_h, можно использовать для вставки в поток перед терминалом, соответствующим пришедшему событию e.

Если лексический анализатор позволяет заглядывать на произвольное количество терминалов вперед, то можно применять оба способа коррекции одновременно, выбирая оптимальный способ в процессе разбора.

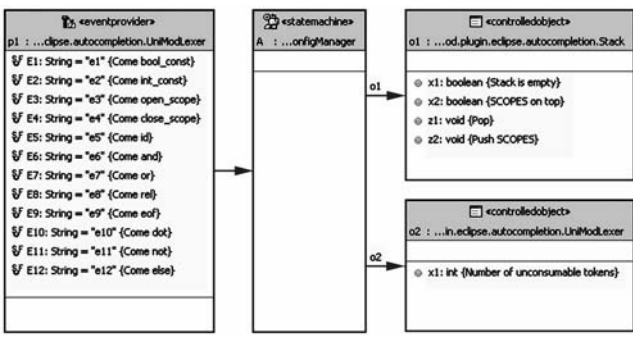
Для выбора оптимального способа авторы предлагают использовать следующее правило:

1) при получении ошибочного терминала в текущем состоянии вычислим количество терминалов, которыми нужно дополнить поток;

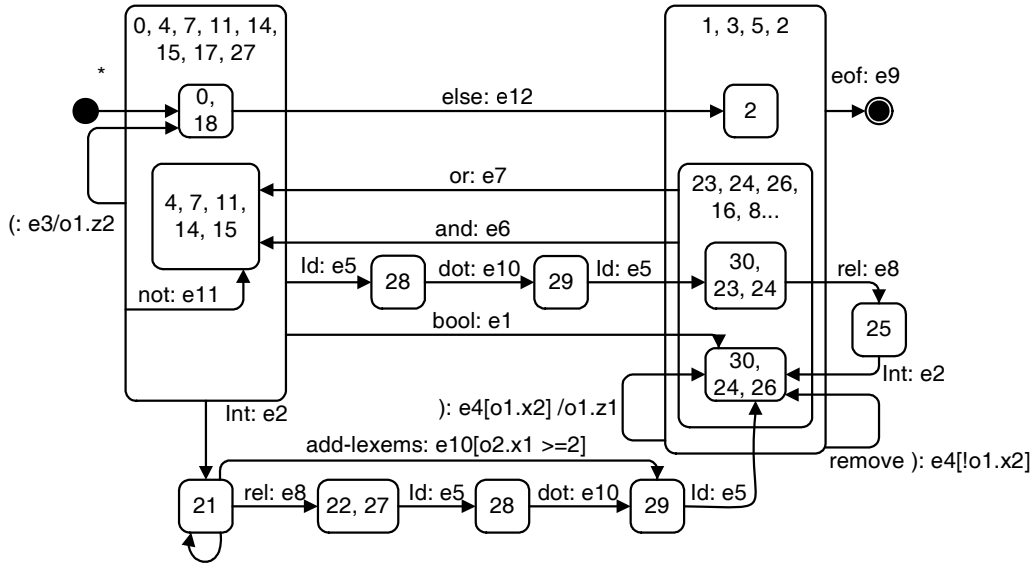
2) вычислим количество терминалов, которое нужно пропустить в потоке, до следующего обрабатываемого в текущем состоянии терминала;

3) выполним коррекцию, требуемое количество терминалов для которой минимально.

Для реализации этого способа коррекции к автомату A распознавателя в качестве объекта управления добавим лексический анализатор (рис. 16).



■ ***Рис. 16.*** *Схема связей модели распознавателя с лексическим анализатором в качестве объекта управления*



■ ***Рис. 17.*** *Добавление переходов, корректирующих поток, в состояние 21*

Лексический анализатор предоставляет автомату распознавателя целочисленную входную переменную *o2.x1*. Ее значение равно числу терминалов, которые необходимо пропустить в потоке, до следующего терминала, обрабатываемого в текущем состоянии. Если входной поток вообще не содержит терминалов, обрабатываемых в текущем состоянии, то значение переменной *o2.x1* больше любого наперед заданного целого числа.

В автомат A добавляются переходы, реализующие и добавление, и пропуск терминалов в потоке. Переходы, реализующие дополнение потока, помечаются следующим условием: длина пути из состояния S в состояние S_h меньше или равна значению входной переменной *o2.x1*. Переходы, реализующие пропуск терминалов, помечаются отрицанием того же условия. Если для состояния S не существует состояния S_h, то переход, удаляющий лексему, выполняется безусловно.

Например, в состоянии *21* нет переходов по событию *e10* — в этом состоянии появление во входном потоке терминала **dot** (точка) не ожидается. Для того чтобы обработать ошибочное появление этого терминала, необходимо добавить два перехода, исходящих из состояния *21* (рис. 17). Ближайшее состояние, в котором обрабатывается событие *e10*, — состояние *28*. Длина пути из состояния *21* в состояние *28* равна двум. Поэтому условие на петле в состоянии *21* по событию *e10* имеет вид $o1.x1 < 2$. Таким образом, в случае, если сразу за терминалом **dot** в потоке следует терминал **rel** (отношение), то терминал **dot** игнорируется. Если следует какой-нибудь другой терминал, то целесообразно сразу перейти в состояние *29*, т. е. добавить в поток отсутствующие терминалы **rel** и **id** (идентификатор).

Описанные выше преобразования могут быть выполнены автоматически для любой диаграммы состояний, так как ближайшее состояние, в котором обрабатывается неожидаемый терминал для данного состояния, можно вычислить, используя, например, алгоритм Флойда—Уоршала [15].

Получение множества строк для автоматического завершения ввода

Предлагаемый алгоритм коррекции входного потока позволяет вычислить множество вариантов завершения как для выражений, являющихся префиксами принадлежащих языку выражений, так и для ошибочных выражений.

После того, как автомат распознавателя, дополненный корректирующими переходами, обработает все терминалы, извлеченные из поданного на вход выражения, он окажется в некотором состоянии S. Для построения множества вариантов завершения следует определить множество переходов, исходящих из S, условия на которых при текущих значениях входных переменных истинны. Терминалы, соответствующие событиям, которыми помечены эти переходы, должны быть преобразованы обратно во множество строк. Например, терминал **id** должен быть преобразован во множество имен переменных, а терминал **and** — в строку «&&». Полученное множество строк и будет множеством вариантов завершения.

Пример работы системы

Приведем пример построения множества вариантов завершения.

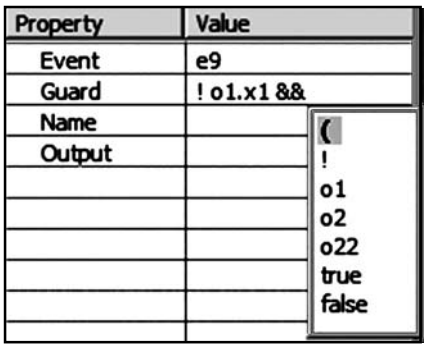
Пусть на вход распознавателю подана строка

! o1.x1 &&.

Лексический анализатор преобразует ее в поток терминалов

not id dot id and,

которому соответствует последовательность событий



■ *Рис. 18. Пример автоматического завершения ввода*

e11, e5, e10, e5, e6.

В процессе обработки этих событий автомат изменяет состояния в следующем порядке:

$$(0,18)\rightarrow(4,7,11,14,15)\rightarrow(28)\rightarrow(29)\rightarrow \rightarrow(30,23,24)\rightarrow(4,7,11,14,15).$$

Состояние (4,7,11,14,15), в котором остановился автомат, содержит исходящие переходы для событий

e1, e2, e3, e5, e11.

Этим событиям соответствуют терминалы

bool, int, '(', id, not,

которые преобразуются в строки:

"true", "false", "(", "o1", "o2", "o22", "!".

Эти строки и формируют множество вариантов завершения для строки, поданной на вход распознавателю.

На рис. 18 показан фрагмент среды разработки с встроенной системой автоматического завершения ввода, описанной в настоящей статье.

Заключение

В работе [5] отмечено, что нерекурсивные нисходящие синтаксические анализаторы можно строить, используя диаграммы состояний, записанные для каждого нетерминала исходной грамматики. Настоящая работа предлагает подход для построения всего одной диаграммы состояний для исходной грамматики. На базе построенной диаграммы реализуется система автоматического завершения ввода. Также отметим, что в известной авторам литературе описание формального метода построения подобных систем отсутствует. Данная работа устраняет указанный пробел.

Реализация системы автоматического завершения ввода для следующей версии проекта UniMod выполнена с помощью предыдущей версии проекта, вследствие чего часть проектной документации была получена «автоматически», так как диаграммы, созданные с помощью UniMod-редактора, являются автоматными программами и могут быть включены в проектную документацию без изменений. Создание последующих версий средств разработки с помощью предыдущих является общепринятой практикой и позволяет говорить о зрелости программного продукта.

Литература

1. **Шалыто А. А., Туккель Н. И.** Танки и автоматы // BYTE/Россия. 2003. № 2. С. 69–73. http://is.ifmo.ru/works/tanks_new/
2. **Шалыто А. А., Туккель Н. И.** SWITCH-технология — автоматный подход к созданию программного обеспечения «реактивных» систем //Программирование. 2001. № 5. С. 45–62. http://is.ifmo.ru/works/switch/1/
3. **Гуров В. С., Мазин М. А., Нарвский А. С., Шалыто А. А.** UML. SWITCH-технология. Eclipse //Информационно-управляющие системы. 2004. № 6. С. 12–17.
4. **Фаулер М.** Рефакторинг. Улучшение существующего кода. М.: Символ-Плюс, 2003. 623 с.
5. **Ахо А., Сети Р., Ульман Д.** Компиляторы: принципы, технологии и инструменты. М.: Вильямс, 2001. 768 с.
6. **Parr T. J., Quong R. W.** ANTRL: A Predicated-LL(k) Parser Generator // Software — Practice And Experience. 1995. N 25 (7). P. 789–810.
7. **Шалыто А. А., Штучкин А. А.** Совместное использование теории построения компиляторов и SWITCH-технологии (на примере построения калькулятора). http://is.ifmo.ru/projects/calc/
8. **Буч Г., Рамбо Г., Якобсон И.** UML. Руководство пользователя. М.: ДМК, 2000. 358 с.
9. **Легалов А. И.** Основы разработки трансляторов. Использование диаграмм Вирта для представления динамически порождаемых конечных автоматов, распознающих КС(1) грамматику. http://softcraft.ru/translat/lect/t08-04.shtml
10. **Шалыто А. А., Туккель Н. И., Шамгунов Н. Н.** Реализация рекурсивных алгоритмов на основе автоматного подхода //Телекоммуникации и информатизация образования. 2002. № 5. С. 72–99. http://is.ifmo.ru/works/recurse/
11. **Хантер Р.** Основные концепции компиляторов. М.: Вильямс. 2002. 256 с.
12. **Акимов О. Е.** Дискретная математика: логика, группы, графы. М.: Лаборатория Базовых Знаний, 2003. 376 с.
13. **Гуров В. С., Нарвский А. С., Шалыто А. А.** Исполняемый UML из России //PC Week/RE. 2005. № 26. С. 18, 19.
14. **UniMod.** http://unimod.sf.net
15. **Кормен Т., Лайзерсон Ч., Ривест Р.** Алгоритмы. Построение и анализ. М.: МЦМНО, 2000. 960 с.