ных технологий в обучении, предложенный профессором Л. А. Растригиным. В качестве модели обучения была взята экспоненциальная зависимость вероятности незнания лексической единицы от скорости и времени забывания.

Ключевые слова: информационно-обучающая технология, состояние памяти модели обучаемого, процесс обучения, порция информации, алгоритм обучения.

УДК 681.3.006

А. С. Кузнецов

МОДЕЛИРОВАНИЕ РАСПОЗНАВАТЕЛЕЙ МУЛЬТИСИНТАКСИЧЕСКИХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ МУЛЬТИВЕРСИОННЫХ СИСТЕМ

Рассмотрено синтезированное абстрактное устройство распознавания мультисинтаксических языков программирования. Это устройство может быть использовано в качестве основы для разработки трансляторов языков программирования мультиверсионных программных систем.

Ключевые слова: мультисинтаксический язык, язык программирования, мультиверсионный подход, программные системы.

Для создания отказоустойчивых программных систем известно множество подходов, использующихся на практике. К их числу относится методология мультиверсионного, или *N*-вариантного программирования [1], одной из отличительных черт которой является разработка нескольких версий одного и того же функционального модуля или компонента. В целом данная методология является дополнением к таким способам получения отказоустойчивых систем, как тестирование и верификация программного обеспечения.

Мультиверсионная программная методология основывается на допустимом разнообразии входных данных, и ее можно ввести в следующие элементы процесса разработки программного обеспечения:

– языки программирования модулей;
– алгоритмы решения типовых задач;
– средства программирования;
– методы тестирования.

Авторами ряда работ по мультиверсионному подходу отмечается тот факт, что при разработке программного обеспечения важно применение различных языков программирования для оформления вариантов [2] или версий [3] программных модулей, поскольку реализация различных версий модуля на разных языках позволяет уменьшить вероятность появления в версиях однотипных ошибок, связанных с языком.

Процесс создания мультиверсионных программных систем подразумевает их архитектурное разбиение на компоненты и модули, создаваемые независимо друг от друга. Однако нередко возникает необходимость в кодировании этих модулей на разных языках в рамках одного и того же исходного текста, как, например при создании динамических HTML-страниц с включением кода на языках JavaScript и (или) VB Script [4]. Традиционный процесс разработки мультиверсионных систем не позволяет выполнить это простым способом. Обойти данное ограничение можно с помощью мультисинтаксических языков программирования [5] и автоматизированной системы MuYacc построения трансляторов языков этого класса.

Итак, в рамках одного исходного текста программы могут одновременно использоваться языки со свойствами, различающимися с точки зрения синтаксиса, и, как следствие, генерируемые разными классами грамматик. В связи с этим такие языки можно отнести к классу мультисинтаксических языков (МСЯ).

При этом синтаксис каждого из составляющих языков описывается контекстно-свободной грамматикой. Из языка-лидера выделяются специальные символы, сигнализирующие главному синтаксическому анализатору о передаче управления соответствующему вспомогательному анализатору. Количество лексических и семантических анализаторов может быть равно количеству используемых языков, а их взаимодействие друг с другом осуществляется по одной из традиционных схем.

Мультиавтомат с магазинной памятью. В основу системы MuYacc положено устройство распознавания МСЯ в виде мультиавтомата с магазинной памятью (МП-мультиавтомат). При этом каждый вспомогательный анализатор основывается на обычном автомате с магазинной памятью (МП-автомат), который принято описывать семеркой элементов [6]:

$$P = \left(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\right),$$

где Q – множество состояний автомата; Σ – алфавит входных символов; Γ – алфавит магазинных символов; δ – графически и таблично заданная функция переходов;

q_0 – начальное состояние автомата; F – множество конечных состояний автомата P; Z_0 – символ, обозначающий дно магазина.

Функция δ с тремя аргументами (q – текущим состоянием автомата, a – текущим входным символом или пустой строкой ε, а также X – символом на вершине магазина) в качестве результата выдает пару элементов (p, γ), где p – новое состояние автомата; γ – цепочка магазинных символов, замещающая X на вершине магазина. Если γ = ε, то магазинный символ снимается. Если γ = X, то содержимое магазина не меняется. Например, если γ = YZ, то X заменяется на Z, а затем Y добавляется в магазин.

Для мультиавтомата P_{mul} данный набор расширяется за счет введения трех дополнительных элементов. Таким образом, требуемое абстрактное устройство описывается следующим образом:

$$P_{\text{mul}} = \left(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F, M, S, \phi\right), \qquad (1)$$

где первые семь элементов имеют тот же смысл, что и в обычном автомате с магазинной памятью; M – множество вспомогательных автоматов; S – особые состояния, соответствующие переключению на какой-либо вспомогательный автомат из множества M; φ – функция переключения на соответствующий вспомогательный автомат из множества M.

Множество $M = \varnothing$ или $M = \bigcup_1^N M_j$, где $M_j, j = \overline{1, N}$ – соответствующий автомат с магазинной памятью из множества мощностью N. Если множество M пустое, то это признак вырожденного мультиавтомата или обычного магазинного автомата. При его программной реализации этот случай не должен восприниматься как ошибочный.

Аналогично множество $S \subseteq Q$ также может быть пустым ($S = \varnothing$). В общем случае $S = \bigcup_1^K S_j$, где $S_j, j = \overline{1, K}$ – особое состояние из соответствующего множества мощностью K. Количество особых состояний может быть больше либо равно количеству вспомогательных автоматов ($K \geq N$), поскольку из различающихся особых состояний возможно переключение на один и тот же автомат.

Мультиавтомат считается заданным ошибочно в следующих случаях:

– если количество особых состояний меньше количества вспомогательных автоматов ($K < N$), так как бессмысленно включать в устройство столь ресурсоемкий элемент, ни разу не получая шанса переключиться на него. Однако при желании это ограничение можно снять, зарезервировав, например, один или несколько вспомогательных автоматов для дальнейшего использования;

– если $M = \varnothing$, а $S \neq \varnothing$, так как тогда исчезает необходимость в выделении особых состояний.

– если $M \neq \varnothing$, а $S = \varnothing$ по тем же соображениям, что и в первом пункте.

Полагая далее, что мультиавтомат не задан ошибочно, опишем функцию переключений φ, которую можно задавать таблично или графически, аналогично функции переходов δ.

Чтобы задать такую функцию, необходимо различать два случая: с наличием в мультиавтомате вспомогательных автоматов и без них.

В первом случае для этого должны быть указаны номер (неотрицательное целое число J_M) вспомогательного магазинного автомата из множества M и номер особого состояния из множества S (J_S). В итоге получаем функцию ϕ_{notempty}, принимающую пять аргументов и дающую на выходе результат в виде набора из трех элементов:

$$\phi_{\text{notempty}}\left(q, a, X, J_M, J_S\right) : \left\{p, \gamma, M_{J_M}\right\}, \qquad (2)$$

где p – новое состояние автомата; γ – цепочка магазинных символов; M_{J_M} – автомат с номером J_M из множества M.

Если взглянуть на это с конструктивной точки зрения, то именно в этот момент и осуществляется переход к вспомогательному автомату с магазинной памятью в виде вызова соответствующей процедуры или перехода по таблице в зависимости от реализации, которая в свою очередь зависит от требуемой эффективности функционирования транслятора.

Теперь предположим, что $M = \varnothing$. Иначе говоря, в нашем абстрактном устройстве нет никаких вспомогательных автоматов. В этом случае необходимо, чтобы функция переключений φ вела себя точно так же, как функция переходов δ обычного автомата. В результате получаем функцию ϕ_{empty}, которая при ближайшем рассмотрении отличается от функции переходов лишь своим названием:

$$\phi_{\text{empty}}\left(q, a, X\right) : \left\{p, \gamma\right\}. \qquad (3)$$

В общем виде функция переключений φ приобретает вид

$$\phi\left(q, a, X, J_M, J_S\right) = \begin{cases} \phi_{\text{notempty}}\left(q, a, X, J_M, J_S\right) : \left\{p, \gamma, M_{J_M}\right\}, \\ \quad \text{если } M \neq \varnothing \text{ и } S \neq \varnothing \text{ и } |M| \leq |S|, \\ \phi_{\text{empty}}\left(q, a, X\right) : \left\{p, \gamma\right\}, \\ \quad \text{если } M = \varnothing \text{ и } S = \varnothing. \end{cases} \qquad (4)$$

Теперь дадим формализованное описание МП-мультиавтомата с учетом выражений (2) и (3):

$$P_{\text{mul}} = \begin{cases} \left(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F, M, S, \phi_{\text{notempty}}\right), \\ \quad \text{если } M \neq \varnothing \text{ и } S \neq \varnothing \text{ и } |M| \leq |S|, \\ \left(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F, M, S, \phi_{\text{empty}}\right), \\ \quad \text{если } M = \varnothing \text{ и } S = \varnothing, \\ \text{Ошибка в противном случае.} \end{cases} \qquad (5)$$

Эквивалентным для (5), с учетом (4), будет следующее формальное определение мультиавтомата, которое отличается от (1) лишь явным указанием на ошибку в описании:

$$P_{\text{mul}} = \begin{cases} \left(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F, M, S, \phi\right), \\ \quad \text{если } |M| \leq |S|, \\ \text{Ошибка в противном случае.} \end{cases} \qquad (6)$$

Функционирование МП-мультиавтомата. Работа простого автомата, а также вырожденного МП-мультиавтомата с магазинной памятью описывается сменой конфигураций [7], где под конфигурацией МП-автомата, или его мгновенным описанием (ID), понимается тройка элементов, описывающих состояние q, оставшуюся часть входной цепочки символов, составляющих программу ω, и содержимое магазина γ. Таким образом $\text{ID} = \left(q, \varpi, \gamma\right)$.

Если P – это МП-автомат, а $\delta(q,a,X)$ содержит (p,α), то $\vdash P$ определяет отношение, которое для всех цепочек ω из Σ^* и β из Γ^* такое, что

$$(q,a\varpi,X\beta) \vdash P\ (p,\varpi,\alpha\beta).$$

По аналогии с этими понятиями дадим определение конфигурации, или мгновенного описания $\mathrm{ID}_{\mathrm{mul}}$, МП-мультиавтомата P_{mul}. Поскольку в некоторый момент времени в выделенном состоянии может быть запущен вспомогательный МП-автомат, то помимо текущего состояния оставшейся части входа и содержимого своего магазина мультиавтомат описывается также через конфигурации каждого из содержащихся в нем МП-автоматов. Последнее является множеством $\mathrm{ID}_{\mathrm{aux}}$ в формуле

$$\mathrm{ID}_{\mathrm{aux}} = \bigcup_{1}^{N} ID_j\ , \qquad (7)$$

где N – количество вспомогательных автоматов. Таким образом, получаем мгновенное описание мультиавтомата в виде следующей формулы:

$$\mathrm{ID}_{\mathrm{mul}} = \begin{cases} (q,\varpi,\gamma,\varnothing), \text{если } P_{\mathrm{mul}} = P, \\ (q,\varpi,\gamma,\mathrm{ID}_{\mathrm{aux}}), \text{в противном случае.} \end{cases} \qquad (8)$$

Также определим для МП-мультиавтомата бинарное отношение перехода $\vdash P_{\mathrm{mul}}$. Если P_{mul} – это описанный МП-мультиавтомат, и $\phi(q,a,X,J_M,J_S)$ содержит (p,α,M_{J_M}), то $\vdash P_{\mathrm{mul}}$ определяет отношение, которое для всех цепочек ω из Σ^* и β из Γ^* такое, что

$$(q,a\varpi,X\beta,M_{J_M}) \vdash P_{\mathrm{mul}}\ (p,\varpi,\alpha\beta,M_{J_M}).$$

Используем символ $\vdash P_{\mathrm{mul}}{}^*$ для представления нуля или более переходов МП-мультиавтомата. Итак, имеем следующее индуктивное определение.

Базис: $I \vdash P_{\mathrm{mul}}{}^* I$ для любого мгновенного описания I.

Индукция: $I \vdash P_{\mathrm{mul}}{}^* J$, если существует некоторое мгновенное описание K, удовлетворяющее условиям

$$I \vdash P_{\mathrm{mul}} K \text{ и } K \vdash P_{\mathrm{mul}}{}^* J.$$

Таким образом, $I \vdash P_{\mathrm{mul}}{}^* J$, если существует такая последовательность мгновенных описаний $K_1, K_2, \ldots, K_n$, у которой $I = K_1, J = K_n$ и $K_i \vdash P_{\mathrm{mul}} K_i + 1$ для всех $I = 1, 2, ..., n-1$.

Языки, допускаемые МП-мультиавтоматами. Получив данное описание абстрактного устройства и механизма его функционирования, мы можем теперь определить через него понятие МСЯ, распознаваемого МП-мультиавтоматом.

Итак, МСЯ является допускаемым мультиавтоматом с магазинной памятью по заключительному состоянию, если он начал свою работу в начальной конфигурации, получив на вход цепочку символов, составляющих программу, прочитал ее полностью, передавая управление соответствующим вспомогательным МП-автоматам (при наличии таковых), и достиг одной из своих конечных конфигураций при условии достижения заключительных конфигураций вспомогательными автоматами или опустошения их магазинов.

Таким образом, если P_{mul} – это мультиавтомат, описанный выражением (6), то тогда языком $L(P_{\mathrm{mul}})$, допускаемым мультиавтоматом по заключительному состоянию, является выражение

$$L(P_{\mathrm{mul}}) = \left\{\omega \middle| (q_0,\omega,Z_0,ID_{\mathrm{mul}}) \vdash P_{\mathrm{mul}}{}^* (q,\varepsilon,\alpha,ID_{\mathrm{mul}})\right\}. \qquad (9)$$

Докажем это утверждение.

1. Сначала предположим, что мультиавтомат не содержит вспомогательных автоматов, тогда, как указывалось выше, он ничем не отличается от обычного МП-автомата. Следовательно, доказательство его сходимости к конечной конфигурации не отличается от доказательства, приведенного, например, в [8].

2. Далее предположим, что рассматриваемый нами мультиавтомат содержит единственный вспомогательный МП-автомат. Поскольку по определению передача ему управления осуществляется в одном из специально выделенных состояний МП-мультиавтомата, а после возврата управления последнему осуществляется переход к следующему обычному состоянию в соответствии с функцией перехода и никакого иного влияния вспомогательный автомат на наше абстрактное устройство не оказывает, то его можно удалить из мультиавтомата. Тогда доказательство сходимости может быть сведено к п. 1.

3. Теперь допустим, что мультиавтомат имеет два вспомогательных МП-автомата. Поскольку, как было указано выше, они друг на друга не влияют, то мы можем поочередно удалить их из мультиавтомата. Тогда доказательство сходимости может быть сведено к п. 2.

4. По этой же схеме доказывается конечность алгоритма работы мультиавтомата при любом количестве МП-автоматов больше одного.

Мультисинтаксический язык является допускаемым мультиавтоматом с магазинной памятью по пустому магазину, если он начал свою работу в начальной конфигурации, получив на вход цепочку символов, составляющих программу, прочитал ее полностью и закончил работу при наличии в магазине лишь маркера его дна:

$$L(P_{\mathrm{mul}}) = \left\{\omega \middle| (q_0,\omega,Z_0,\mathrm{ID}_{\mathrm{mul}}) \vdash P_{\mathrm{mul}}{}^* (q,\varepsilon,\varepsilon,\mathrm{ID}_{\mathrm{mul}})\right\}. \qquad (10)$$

Ограничения на вспомогательные автоматы здесь те же, что и в выражении (9).

В [8] приводится доказательство эквивалентности языков, допускаемых по пустому магазину и по заключительному состоянию. Следовательно, построив мультиавтомат с магазинной памятью, допускающий язык по пустому магазину, эквивалентный такому же устройству, допускающему язык по заключительному состоянию, мы можем использовать доказательство выражения (9).

Формирование таблиц синтаксического анализа мультисинтаксических языков программирования. Итак, предложенный выше МП-мультиавтомат может быть использован в качестве основы программы синтаксического анализа МСЯ. Однако на каждом шаге он может требовать наличия нескольких магазинов произвольного размера в каждый момент времени, что не всегда приемлемо даже для ресурсов современных средств вычислительной техники. В силу этой и ряда других причин, трансляторы языков программирования, как правило, строятся на базе таблично-управляемых алгоритмов синтаксического анализа [7]. Эти алгоритмы в достаточной мере представлены в специальной литературе, и автором данной статьи было принято решение модифицировать один из алгоритмов формирования таблиц синтаксического

анализа с тем, чтобы в дальнейшем воспользоваться им для создания системы MuYacc.

Известно, что метод LALR(1) восходящего синтаксического анализа позволяет покрыть большинство синтаксических аспектов применяемых на практике языков программирования [7]. Модификация этого метода позволила получить алгоритм mLALR (1), который унаследовал от своего родителя необходимость построения таблиц, строки которых соответствуют состояниям синтаксического анализатора, а столбцы – терминальным и нетерминальным символам контекстно-свободной грамматики, описывающей каждый язык-компонент МСЯ. При этом для анализаторов вспомогательных языков используется классический LALR(1)-алгоритм с элементами, которые могут принимать значения четырех типов [7]. В то же время синтаксический анализатор для языка-лидера использует таблицы с элементами пяти типов, из которых первые четыре унаследованы от LALR(1)-алгоритма:

– элементы переноса. Эти элементы имеют вид S_k и означают следующее: поместить в магазин символ, соответствующий столбцу; поместить в магазин состояние k и перейти к состоянию k; если рассматриваемый символ – терминальный, то принять его. Если данный перенос подразумевает запуск анализатора вспомогательного языка, то это действие выполняется;

– элементы свертки. Они имеют вид R_m и означают, что нужно выполнить свертку с помощью правила m; удалить N символов и N состояний из магазина (N – количество символов в правой части правила m); перейти к состоянию, находящемуся на вершине магазина. Нетерминал из левой части продукции m считается следующим входным символом;

– элементы ошибок. Они являются пустыми ячейками таблицы и соответствуют синтаксическим ошибкам;

– элементы остановки, которые имеют вид Accept. Ими завершается процесс синтаксического анализа (входная строка принимается). Если анализируется вспомогательный язык, то процедуре анализа лидирующего языка возвращается успешное значение;

– элементы запускающего переноса. Эти элементы имеют вид E_k и означают следующее: поместить в магазин символ, соответствующий столбцу; поместить в магазин состояние k; найти и запустить анализатор соответствующего вспомогательного языка; если этот анализатор закончил анализ состоянием ошибки, то завершить анализ лидирующего языка с ошибкой; в противном случае перейти к состоянию k. Поскольку рассматриваемый символ – терминальный, то его принимают.

Анализатор языка-лидера начинает свою работу из начального состояния 0 и, переходя из одного состояния в другое, может закончить свою работу либо в состоянии Accept, либо в состоянии ошибки.

Таким образом, под руководством автора данной статьи был разработан компилятор компиляторов MuYacc [9], предназначенный для автоматизации процесса генерации трансляторов МСЯ. В качестве базиса MyYacc выбран описанный выше мультиавтомат с магазинной памятью. Это программное средство использовалось для создания экспериментального компилятора языка программирования С с возможностью включения кода на языке ассемблера архитектуры x86 в двух известных нотациях – Intel и AT&T. Опытная эксплуатация полученного компилятора показала практически двукратное снижение количества ошибок при построении второй версии двухвариантных модулей общим размером около 8 KLOC независимо от порядка их конструирования.

Библиографический список

1. Avizienis, A. The Methodology of N-Version Programming / A. Avizienis // Software Fault Tolerance. New York : John Wiley & Sons, 1995. P. 23–47.

2. Горбунов-Посадов, М. М. Расширяемые программы / М. М. Горбунов-Посадов. М. : Полиптих, 1999.

3. Соммервилл, И. Инженерия программного обеспечения / И. Соммервилл. М. : Вильямс, 2002.

4. Глушаков, С. В. Программирование Web-страниц. JavaScript. VBScript / С. В. Глушаков, И. А. Жакин, Т. С. Хачиров. Харьков : Фолио, 2002.

5. Кузнецов, А. С. Автоматизация процесса генерации компиляторов мультисинтаксических языков программирования / А. С. Кузнецов, И. В. Ковалев, Е. А. Веретенников // Вестник СибГАУ. 2007. Вып. 3(16). С. 73–75.

6. Карпов, Ю. Г. Теория и технология программирования. Основы построения трансляторов / Ю. Г. Карпов. СПб. : БХВ-Петербург, 2005.

7. Ахо, А. Компиляторы: принципы, технологии и инструментарий / А. Ахо, М. Лам, Р. Сети, Дж. Ульман. 2-е изд. М. : Вильямс, 2008.

8. Хопкрофт, Дж. Э. Введение в теорию автоматов, языков и вычислений / Дж. Э. Хопкрофт, Р. Мотвани, Дж. Д. Ульман. 2-е изд. М. : Вильямс, 2002.

9. Кузнецов, А. С. Генерация компиляторов мультисинтаксических языков программирования мультиверсионных систем / А. С. Кузнецов, И. В. Ковалев // Прогр. продукты и системы. 2008. Вып. 4 (84). С. 101–103.

A. S. Kuznetsov

MODELING FOR RECOGNIZERS OF MULTISYNTAX PROGRAMING LANGUAGES FOR MULTI VERSION SYSTEMS

The abstract device for recognizing of multi syntax programming languages to construct multi version programming systems is covered. The synthesized device is a complex can be used as a base for programming language translation tool of multi version programming system development.

Keywords: multi syntax language, programming language, multi version approach, program system.