УДК 681.3.068

ФОРМИРОВАНИЕ ТАБЛИЦ СИНТАКСИЧЕСКОГО АНАЛИЗА МУЛЬТИСИНТАКСИЧЕСКИХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Кузнецов А.С.

Научный руководитель – профессор Ковалев И.В.

Сибирский федеральный университет

В рамках мультиверсионной методологии разработки отказоустойчивого программного обеспечения выполняется диверсификация на уровне используемых языков программирования, а один из возможных вариантов – использование мультисинтаксических языков. Язык данного типа может состоять из языка-*лидера* и одного или более *вспомогательных* языков. Примерами последних являются ассемблерные вставки в код на языке высокого уровня, среди которых можно выделить C/C++ и Smalltalk. Несколько версий одного модуля и функции в этом случае строятся на нескольких языках с различающимся синтаксисом и одной общей семантической базой.

Анализ мультисинтаксических языков в общем случае требует нескольких совместно функционирующих распознавателей. Распознаватели мультисинтаксических языков программирования, например, мультиавтоматы с магазинной памятью или автоматы с двумя и более магазинами, для корректного функционирования на каждом такте могут требовать наличия нескольких стеков произвольного. Это не всегда приемлемо даже для ресурсов современных вычислительных средств. По ряду причин трансляторы промышленных языков программирования, как правило, строятся на базе таблично-управляемых алгоритмов синтаксического анализа. Эти алгоритмы в достаточной мере освещены в специальной литературе, и автором было принято решение о модификации одного известного алгоритма формирования таблиц синтаксического анализа. На его основе можно реализовать необходимые инструментальные средств разработки трансляторов.

Синтаксические анализаторы, разрабатываемые с использованием инструментов семейства уасс, функционируют по алгоритму LALR(1), который осуществляет предпросмотр одного символа анализируемой цепочки для разрешения конфликтов, которые потенциально могут возникнуть при расстановке действий в ячейках таблиц.

Как известно, LALR(1)-анализ позволяет покрыть большинство синтаксических аспектов применяемых на практике языков программирования. Кроме того, такой известный генератор компиляторов, как bison, допускает использование более общего алгоритма GLR(1)-анализа, модификацию которого автор предполагает осуществить в будущем наряду с некоторыми другими, в частности возвратными методами восходящего синтаксического анализа, а также методами, использующими переменное число символов предпросмотра.

В модифицированном алгоритме LALR(1)-анализа, который назван mLALR(1)-алгоритмом *m* расшифровывается как «мультисинтаксический», а оставшиеся знаки сохранили свое смысловое значение. Полное описание метода LALR(1) дано, например, в работе Ахо, Сети и Ульмана «Компиляторы: принципы, технологии, инструменты», широко известной как «Книга Красного Дракона».

Единственное, но существенное отличие этого алгоритма от исходного заключается в запуске вспомогательного анализатора, когда распознается первый символ правой части продукции, называемый *особой лексемой*, вслед за которой идет такой грамматический символ, который и представляет собой включение кода на одном языке в программу на другом языке. Поскольку этот символ является терминальным и распознается лексическим анализатором или аналогичной фазой процесса трансляции, то в такой ситуации должно выполняться действие переноса. По этой причине такое «гибридное» действие назовем *запускающим переносом* (*executive shift*).

Классический алгоритм LR(1)-анализа, а также алгоритм LALR(1) являются восходящими, то есть строят дерево разбора от листьев к корню. Двумя основными действиями построенного по этим алгоритмам синтаксического анализатора являются перенос и свертка.

Перенос (*Shift*) соответствует продвижению на один пункт по какому-либо правилу грамматики. Переносы выполняются до тех пор, пока не достигнут конца рассматриваемой продукции. При этом активно используется такая структура как автомат с магазинной памятью, в которой размещаются элементы частично распознанных продукций и все необходимые предыдущие состояния синтаксического анализатора.

Свертка (*Reduce*) соответствует достижению конца рассматриваемой продукции. В результате из магазина будут извлекаться N символов и N сохраненных состояний, где N – это количество символов правой части продукции.

В детерминированных методах восходящего синтаксического анализа единственная информация, которой должен владеть синтаксический анализатор – какое действие требуется выполнить в конкретный момент времени (например, на соответствующем такте мультиавтомата с магазинной памятью). Принятие решения по этому вопросу основывается на таблицах синтаксического анализа, в которых столбцам соответствуют – терминальные и нетерминальные символы, а также *маркер конца строки*. Строкам соответствуют все возможные состояния анализатора.

С учетом того, что анализируется мультисинтаксический язык программирования, составленный из нескольких языков, то требуется использовать не одну, а несколько таблиц синтаксического анализа. Для языков, названных выше вспомогательными, таблица состоит из элементов четырех типов, которые эквивалентны аналогичным действиям алгоритма LALR(1):

1. *Элементы переноса*. Эти элементы имеют вид S_k и означают: поместить в магазин символ, соответствующий столбцу; поместить в магазин состояние k и перейти к состоянию k; если рассматриваемый символ – терминальный, то принять его.

2. *Элементы свертки*. Они имеют вид R_m и означают: выполнить свертку с помощью правила m; удалить N символов и N состояний из магазина (N – это количество символов в правой части продукции с номером m); перейти к состоянию, находящемуся на вершине магазина. Нетерминал из левой части продукции m считать следующим входным символом.

3. *Элементы ошибок*. Они являются пустыми ячейками таблицы и соответствуют синтаксическим ошибкам.

4. *Элементы остановки*, которые имеют вид *Accept*. Ими завершается процесс синтаксического анализа (входная строка, или программа на мультисинтаксическом языке, *принимается*). Если анализируется вспомогательный язык, то возвратить процедуре анализа лидирующего языка успешное значение.

Для **языка-лидера** в одной или нескольких позициях таблицы синтаксического анализа также могут быть размещены элементы пятого типа, а именно:

5. *Элементы запускающего переноса*. Эти элементы имеют вид E_k и означают: 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 принять его.

Таким образом, важнейшей частью алгоритма синтаксического анализа языка-лидера является этап расстановки действий в ячейки таблиц синтаксического анализа. Однако перед этим должны быть сформированы две таблицы:

- O особых лексем;

- P процедур синтаксического анализа вспомогательного языка.

Сохраняя смысл понятий LR(1)-ситуаций, множеств ситуаций и ядер множеств, символов предпросмотра, а также действий *action* и *goto*, используемых в алгоритме LALR(1), дальнейшие действия будут следующими.

Алгоритм построения таблиц синтаксического mLALR(1)-анализа мультисинтаксических языков программирования.

Шаг 0. Строятся таблицы O особых лексем и P процедур синтаксического анализа вспомогательных языков.

Шаг 1. Строится $C = \{I_0, I_1, \ldots, I_n\}$ – система множеств LR(1)-ситуаций для грамматики языка-лидера.

Шаг 2. Для каждого ядра, имеющегося среди множества LR(1)-ситуаций, осуществляется поиск всех множеств с этим ядром, а затем этим множества заменяются их объединением. В результате будут получены C’ = {J_0, J_1, …, J_m} – множества LR(1)-ситуаций.

Шаг 3. На основе J_i строится состояние i. Действия синтаксического анализатора для данного состояния определяются так:

а) если $[A \to \alpha \cdot a\beta, b] \in J_i$, и $goto(J_i, a) = J_l$, и $a \in O$, то *action*[i, a] ставится равным E_l. (запускающий перенос).

б) если $[A \to \alpha \cdot a\beta, b] \in J_i$, и $goto(J_i, a) = J_l$, и $a \notin O$, то *action*[i, a] ставится равным S_l (перенос).

в) если $[A \to \alpha \cdot, a] \in J_i$, $A \neq S'$, то *action*[i, a] ставится равным R_m (свертка $A \to \alpha$).

г) если $[S' \to S \cdot, \$] \in J_i$, то *action*[$i$, $\$$] ставится равным *Accept* (допуск).

Если соблюдение этих правил приводит к конфликту, то грамматика не является LALR(1) и данный алгоритм к ней неприменим.

Шаг 4. Строится таблица *goto*. Если J – объединение множеств LR(1)-ситуаций, то $J = I_1 \cup I_2 \cup \ldots \cup I_k$, то ядра множеств *goto*(I_1, X), …, *goto*(I_k, X) одни и те же, поскольку I_1, …, I_k имеют одно и то же ядро. Обозначим через K объединение всех множеств ситуаций, имеющих то же ядро, что и *goto*(I_1, X). Тогда можно получить, *goto*(J, X) = K.

Шаг 5. Все записи, которые не определены правилами (3) и (4), задаются как «ошибка», которой на практике соответствует пустая ячейка.

Шаг 6. Начальное состояние анализатора представляет собой состояние, построенное из множества, содержащего ситуацию $[S' \to \cdot S, \$]$.

После выполнения всех шагов данного алгоритма можно начать функционирование mLALR(1)-анализа как таковое. Оно будет заключаться в выполнении действий, полученных в ходе формирования согласно описанному выше алгоритму и записанных в таблицах синтаксического анализа, включая позиции синтаксических ошибок.