Московский авиационный институт

(национальный исследовательский университет)

Факультет «Компьютерные науки и прикладная математика»

**Курсовая работа**

**по курсу**

**«Системы программирования»**

**IV семестр**

**«**Реализация алгоритма построения LR(k) анализатора на основе активных префиксов и отношения OBLOW**»**

*Студент:* Боташев А.Р.

*Группа:*  М8О-201Б-21

*Руководитель:* Киндинова В. В.

*Оценка:*

*Дата:*

**Москва, 2023**

**LR(k) анализатор**

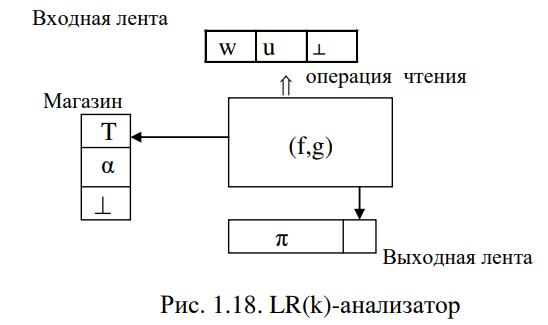
КС - грамматика G = (T, V, P, S) является **LR(k)-грамматикой**, если просмотрев только часть кроны дерева вывода в этой грамматике, расположенной слева от данной внутренней вершины, часть кроны, выведенную из нее, и следующие k символов входной цепочки, можно установить правило вывода, которое было применено к этой вершине при порождении входной цепочки.

Грамматики, для которых можно построить детерминированный восходящий анализатор, называются LR(k)-грамматиками (входная цепочка читается слева (Left) направо, выходом анализатора является правый (Right) разбор, k-число символов входной цепочки, на которое можно “заглянуть” вперед для выделения основы).

Системы, распознающие языки, порожденные подобной грамматикой, называются **LR-анализаторами**.

Если в процессе вывода цепочки правила грамматики применяются только к самому правому нетерминалу, говорят, что получен **правый вывод цепочки**.

Синтаксический LR-анализатор анализирует входную цепочку слева направо (L), и строит правый (R) вывод грамматики.



**Активные префиксы и отношение OBLOW**

**Грамматическое вхождение** – это символы полного словаря грамматики, снабженные двумя индексами. Первый индекс i задает номер правила грамматики, в правую часть которого входит данный символ, а второй индекс j – номер позиции символа в этой правой части.

Перед тем, как определить отношение OBLOW, определим множество OFIRST(u). Определим множество OFIRST(Yi) (входит первым), в которое включим Yj и все грамматические вхождения, с которых могут начинаться цепочки, выводимые из Y:

OFIRST(Yi) = {Yj} U {Xi | Y =>\* Aβ => Xαβ и Xi самое левое

грамматическое вхождение в правую часть правила A → X}

Множество OFIRST(Yi)

1. **private** List<Symbol> OFirst(Symbol occur) // Возвращает множество OFIRST для грам. вхождения
2. {
3. var result = **new** List<Symbol>();
4. result.Add(occur);
5. **foreach** (var p **in** grammar.P) {
6. var rhs = p.RHS;
7. **if** (p.LHS == occur.symbol) {
8. **if** (occur.symbol == rhs[0]) {
9. result.Add(**new** Symbol(rhs[0].symbol, p.Id, 1));
10. } **else** {
11. var newResult = OFirst(**new** Symbol(rhs[0].symbol, p.Id, 1));
12. result.AddRange(newResult);
13. }
14. }
15. }
16. **return** result;
17. }

Определим отношение OBLOW (входит под) используя множество OFIRST, следующим образом:

Xi OBLOW Yj - это множество {(Xi, Yj) | A→ αXiZjβP и Yj∈OFIRST(Zj)}

OBLOW Yj - это множество {(⟂, Yj) | Yj ∈ OFIRST(S0)}, где S0 -

начальное вхождение.

Отношение Xi OBLOW Yj определяет множество Q грамматических вхождений Xi, для которых представляющие их магазинные символы могут встретиться в магазине непосредственно под символом, представляющим Yj.

Отношение OBLOW будем представлять в виде матрицы, содержащей n столбцов и (n+1) рядов, где n - число грамматических вхождений пополненной грамматики G. Первые n рядов матрицы отмечены грамматическими вхождениями, а последний ряд - маркером дна стека. Если Xi OBLOW Yj, то элемент матрицы, расположенный в ряде Xi и столбце Yj равен 1.

1. // Нахождение матрицы отношения OBLOW и множества грамматических вхождений
2. var oblow = **new** Dictionary<Symbol, Dictionary<Symbol, **int**>>();
3. var allGrammarOccurs = **new** HashSet<Symbol>();
4. **foreach** (var p **in** grammar.P)
5. {
6. var rhs = p.RHS;
7. **if** (p.LHS == "П")
8. {
9. var symbol = **new** Symbol(rhs[0].ToString(), p.Id, 1);
10. var botMarker = **new** Symbol("^", 0, 0);
11. allGrammarOccurs.Add(botMarker);
12. allGrammarOccurs.Add(symbol);
13. var first = OFirst(symbol);
14. oblow.Add(botMarker, **new** Dictionary<Symbol, **int**>());
15. **foreach** (var occur **in** first)
16. {
17. oblow[botMarker].Add(occur, 1);
18. }
19. }
20. **else**
21. {
22. **for** (**int** j = 0; j < rhs.Count; ++j)
23. {
24. var current = **new** Symbol(rhs[j].ToString(), p.Id, j+1);
25. allGrammarOccurs.Add(current);
26. **if** (j != rhs.Count - 1)
27. {
28. var first = OFirst(**new** Symbol(rhs[j + 1].ToString(), p.Id, j+2));
29. oblow.Add(current, **new** Dictionary<Symbol, **int**>());
30. **foreach** (var occur **in** first) oblow[current].Add(occur, 1);
31. }
32. }
33. }
34. }

Выполнение алгоритма описывается в терминах конфигураций, представляющих собой тройки вида (αT, ax, π), где αT – цепочка магазинных символов (Т-верхний символ магазина), ax – необработанная часть входной цепочки, π выход (строка из номеров правил), построенный к настоящему моменту времени.

Таблица состоит из двух подтаблиц – функции действия и функции переходов.

Входным символам с ленты соответствуют столбцы таблицы, символам магазина – строки.

Управление алгоритмом осуществляется двумя функциями, задаваемых в таблице:

1. По верхнему символу магазина и входному символу, определяется функция действия: ПЕРЕНОС или СВЕРТКА;

2. ПЕРЕНОС определяет значение функции перехода, равное магазинному символу, который нужно втолкнуть в магазин;

3. СВЕРТКА(i), однозначно определяет правило i для свертки.

Кодируемому магазинному символу соответствует единственное вхождение (детерменированный КА).

Необходимо определить функцию действий F(u) на множестве (Vp ∪ {×(T ∪ {ε}) по следующим правилам:

1. Если A → β – правило грамматики с номером i, то для конфигурации (αT, ax, π), где T кодирует цепочку β, F(u) = С(i) = (СВЕРТКА, i).

Если магазинному символу u соответствует только одно грамматическое вхождение Xi, являющееся самым правым вхождением в i-е правило вывода грамматики G, то все элементы ряда, помеченной г, имеют значение (СВЕРТКА, i).

2. Если A → β – правило грамматики с номером i, то для конфигурации (αT, ax, π), где u кодирует некий префикс цепочки β (но не саму основу), F(u) = Перенос.

3. Для конфигурации (S0, π), где S0 кодирует цепочку S, F(u) = ДОПУСК.

4. В противном случае, F(u) = ОШИБКА.

Рассмотрим на примере:

Исходная грамматика:

G = (**T**, **V**, **P**, **S**), где

**T** = {a, c, d, k}, **V** = {S, A, B, C, D, V, V’}, **S0** = S

**P**:

p1: S → AB

p2: A → CD

p3: B → a

p4: B → dV

p5: C → d

p6: D → c

p7: V → k

p8: V → kV’

p9: V’ → k

p10: V’ → kV’

Построим матрицу отношения OBLOW

| **OBLOW** | S0 = S | A1 | B1 | C2 | D2 | a3 | d4 | V4 | d5 | c6 | k7 | k8 | V’8 | k9 | k10 | V’10 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S0 = S |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| A1 |  |  | 1 |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  |
| B1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| C2 |  |  |  |  | 1 |  |  |  |  | 1 |  |  |  |  |  |  |
| D2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| a3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| d4 |  |  |  |  |  |  |  | 1 |  |  | 1 | 1 |  |  |  |  |
| V4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| d5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| c6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| k7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| k8 |  |  |  |  |  |  |  |  |  |  |  |  | 1 | 1 | 1 |  |
| V’8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| k9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| k10 |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 | 1 | 1 |
| V’10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ⊥ | 1 | 1 |  | 1 |  |  |  |  | 1 |  |  |  |  |  |  |  |

Если A OBLOW B, то на пересечении строки A со столбцом B пишем 1.

Теперь на основе правил заполнения, построим:

| Символ грамматики | Магазинный символ | Кодируемая цепочка | Операции |
| --- | --- | --- | --- |
| S0 =S | S0 | S | Д |
| A | A1 | A | П |
| B | B1 | AB | C1 |
| C | C2 | C | П |
| D | D2 | CD | С2 |
| V | V4 | dV | C4 |
| V’ | V’8  V’10 | kV’  kV’ | C8  C10 |
| a | a3 | a | C3 |
| c | C6 | c | C6 |
| d | d5 | d | C5 |
| k | k7  k8  k9  k10 | k  k  k  k | C7  П  C8  П |

Для каждого магазинного символа (за исключением S0 и ┴), кодируемая цепочка является префиксом правой части некоторого правила грамматики.

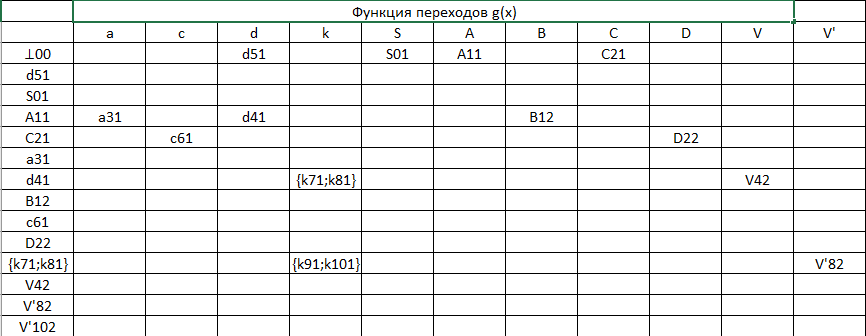
Индекс каждого символа соответствует номеру правила, префикс правой части которого кодируется этим символом. И наоборот, каждый непустой префикс правой части правила кодируется некоторым магазинным символом.

Символ переносится в магазин только в том случае, если он кодирует цепочку, «совместимую» с цепочкой, которая будет находиться в магазине после переноса.

Цепочка, кодируемая данным магазинным символом, совместима с цепочкой в магазине, если она является суффиксом магазинной цепочки после переноса данного символа.

1. // Построение промежуточной матрицы для таблицы переходов g(X)
2. var building = **new** Dictionary<Symbol, Dictionary<Symbol, List<Symbol>>>();
3. HashSet<Symbol> nonDeterministic = **new** HashSet<Symbol>();
5. **foreach** (var x **in** allGrammarOccurs) {
6. building.Add(x, **new** Dictionary<Symbol, List<Symbol>>());
7. **foreach** (var y **in** allGrammarOccurs) {
8. **if** (y != "^" && oblow.ContainsKey(x) && oblow[x].ContainsKey(y)) {
9. **if** (!building[x].ContainsKey(y.symbol))
10. building[x].Add(y.symbol, **new** List<Symbol>());
11. building[x][y.symbol].Add(y);
12. **if** (building[x][y.symbol].Count > 1) {
13. **foreach** (var bad **in** building[x][y.symbol]) {
14. nonDeterministic.Add(bad);
15. }
16. }
17. }
18. }
19. }

Построим таблицу переходов(с помощью матрицы отношения OBLOW):



Для построения таблицы функции Перенос-Свертка для каждого правила необходимо:

* Если символ кодирует всю правую часть правила (то есть символ является самым правым) - значение действия в этом символе - СВЕРТКА
* Если символ кодирует не всю правую часть правила (то есть символ не является самым правым) - значение действия в этом символе - ПЕРЕНОС
* На пересечении S0 и ┴ значение функции действия - ДОПУСК.
* Во всех незаполненных ячейках значение функции действия - ОШИБКА

Рассмотрим пример заполнения функции действий перенос-свертка на примере текущей грамматики:

p1: S → AB

p2: A → CD

p3: B → a

p4: B → dV

p5: C → d

p6: D → c

p7: V → k

p8: V → kV’

p9: V’ → k

p10: V’ → kV’

Изначально поставим на пересечении S0 и ┴ значение ДОПУСК.

Будем перебирать правила, в правых частях которых содержится терминал.

*Для правила 3*:

Если мы встретим на входной ленте терминал “a” и на вершине магазина будет находится символ “А1” - нужно сделать перенос “а” в магазин (смотри правило 1). Значит, на пересечении “A1” и “a” ставим значение ПЕРЕНОС.

Если мы встретим на входной ленте терминал “a” и на вершине магазина будет находится любой из символов, стоящий последним в любой из выведенных цепочек из “А1” - значит нам нужно сделать свертку(смотри правило 1). На пересечении “c6” и “а” ставим значение СВЕРТКА6, на пересечении “D6” и “a” ставим значение СВЕРТКА2

*Для правила 4*:

Если мы встретим на входной ленте терминал “d” и на вершине магазина будет находится символ “А1” - значит нам нужно сделать перенос “d” в магазин (смотри правило 1).

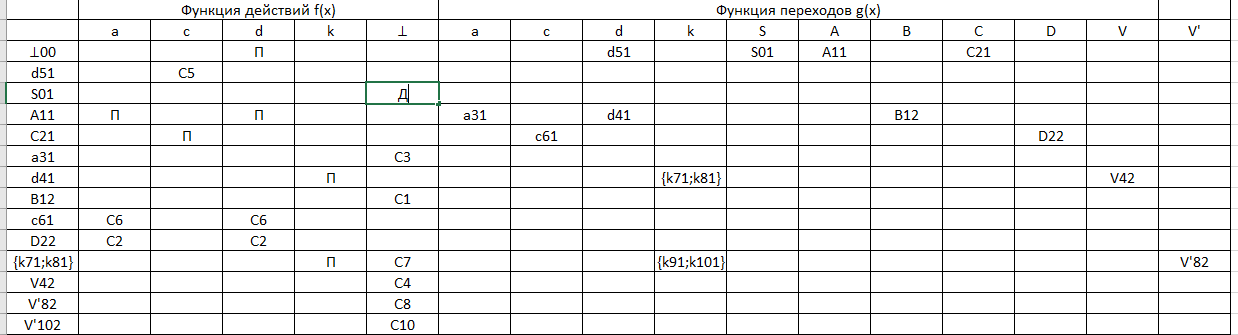
Если мы встретим на входной ленте терминал “d” и на вершине магазина будет находится символ “с6” или “D6” - нам необходимо сделать свертку(аналогично правилу 3)

*Для правила 5*:

если мы встретим на входной ленте терминал “d” и на вершине магазина будет находится маркет дна - значит нам нужно сделать перенос “d” в магазин (смотри правило 1).

Аналогично заполняем всю таблицу f(x) и объединяем ее с таблицей переходов.

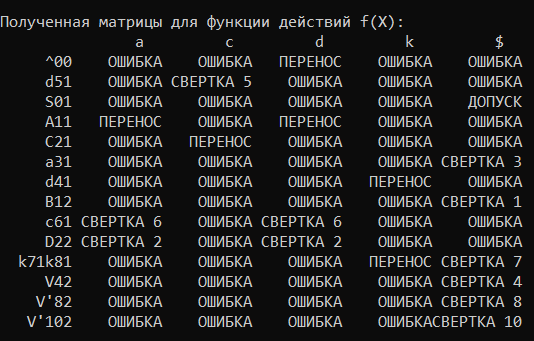
Во всех незаполненных ячейках таблицы ставим значение ОШИБКА



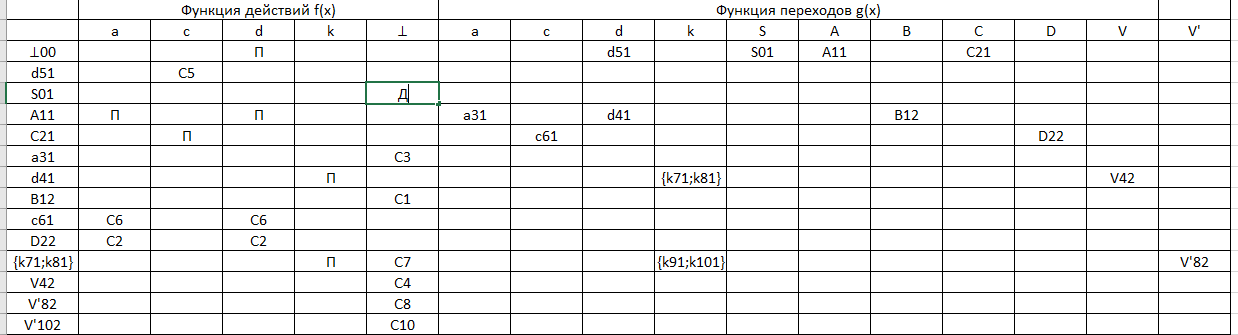
Наконец, можно убедиться в корректности ручного заполнения таблицы с помощью программы:

1. var Z = **new** Dictionary<**string**, Dictionary<Symbol, **string**>>();
2. var Vp = **new** HashSet<**string**>();
3. var M = **new** Dictionary<**string**, List<Symbol>>();
5. var alphabet = **new** List<Symbol>(grammar.T);
6. alphabet.AddRange(grammar.V);
7. **foreach** (var x **in** allGrammarOccurs) {
8. **if** (nonDeterministic.Contains(x))
9. **continue**;
10. var XArray = **new** List<Symbol>();
11. XArray.Add(x);
12. **string** vX = GetVp(XArray, M);
13. Vp.Add(vX);
14. Z.Add(vX, **new** Dictionary<Symbol, **string**>());
15. **foreach** (var t1 **in** alphabet) {
16. **if** (t1 == "$" || t1 == "П")
17. **continue**;
18. **if** (!building.ContainsKey(x) || !building[x].ContainsKey(t1))
19. **continue**;
20. **string** vD = GetVp(building[x][t1], M);
21. Vp.Add(vD);
22. Z[vX].Add(t1, vD);
23. **if** (building[x][t1].Count > 1 && !Z.ContainsKey(vD)) //добавлен if
24. {
25. Z.Add(vD, **new** Dictionary<Symbol, **string**>());
26. **foreach** (var d **in** building[x][t1]) {
27. **foreach** (var t2 **in** alphabet) {
28. **if** (building.ContainsKey(d) && building[d].ContainsKey(t2)) {
29. Z[vD].Add(t2, GetVp(building[d][t2], M));
30. }
31. }
32. }
33. }
34. }
35. }





Видим, что программное и ручное заполнение совпадают.



Подробнее рассмотрим процесс работы LR(k) анализатора.

Пусть на входной ленте дана цепочка dcdkkk. Изначальная конфигурация автомата:

(⊥, dcdk⊥, ɛ)

Считывающая головка стоит на символе d, на вершине стека магазина находится маркер дна. Смотрим пересечение маркера дна и символа d в таблице функции Перенос-Свертка - указано действие ПЕРЕНОС. Следовательно, переносим символ d c входной ленты на вершину стека магазина. Для определения индекса d смотрим в таблицу функции переходов на пересечении ⊥ и символа d - там указан символ d с индексом 5. Значит, на вершину магазина переносится d5

(⊥d5, cdk⊥, ɛ)

Cчитывающая головка на c, на вершине магазина d5. Определяем действие по функции действий - СВЕРТКА5. Смотрим на правило 5: C -> d. Определяем индекс, с которым мы должны свернуть терминал в нетерминал C с помощью функции переходов и делаем свертку: заменяем d5 на вершине стека на C2, затем записываем на выходную ленту номер правила, по которому произошла свертка:

(⊥C2, cdk⊥, 5)

Cчитывающая головка на c, на вершине магазина C2. Действие по таблице действий - ПЕРЕНОС. Индекс по таблице переходов - 6.

(⊥C2с6, dk⊥, 5)

Cчитывающая головка на d, на вершине магазина с6. Действие по таблице действий - СВЕРТКА6. Индекс по таблице переходов - 2.

(⊥C2D2, dk⊥, 56)

Cчитывающая головка на d, на вершине магазина D2. Действие по таблице действий - СВЕРТКА2. Индекс по таблице переходов - 2.

(⊥A1, dk⊥, 562)

Cчитывающая головка на d, на вершине магазина A1. Действие по таблице действий - ПЕРЕНОС. Индекс по таблице переходов - 4.

(⊥A1d4, k⊥, 562)

Cчитывающая головка на d4, на вершине магазина k. Действие по таблице действий - ПЕРЕНОС. Индекс по таблице переходов - 4.

(⊥A1d4k7, ⊥, 562)

Cчитывающая головка на ⊥, на вершине магазина k7. Действие по таблице действий - СВЕРТКА7. Индекс по таблице переходов - 4.

(⊥A1d4V4, ⊥, 5627)

Cчитывающая головка на ⊥, на вершине магазина V4. Действие по таблице действий - СВЕРТКА4. Индекс по таблице переходов - 1.

(⊥A1B1, ⊥, 56274)

Cчитывающая головка на ⊥, на вершине магазина V4. Действие по таблице действий - СВЕРТКА4. Индекс по таблице переходов - 1.

(⊥S0, ⊥, 562741)

Cчитывающая головка на ⊥, на вершине магазина V4. Действие по таблице - ДОПУСК. Анализатор принял строку.