# Теория автоматов и формальных языков

## Лабораторная работа №6

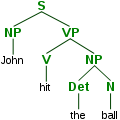
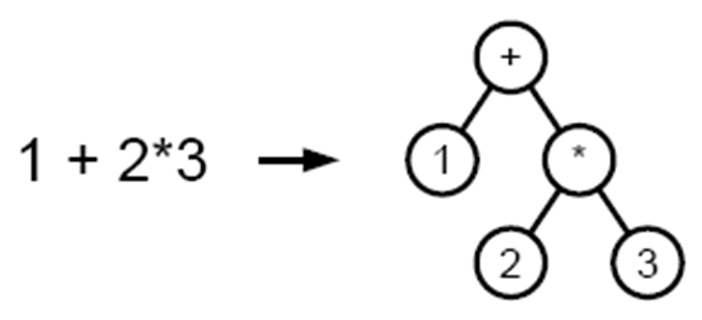
# Создание синтаксического анализатора

**Синтаксический анализ** (синтаксический разбор, парсинг) – процесс сопоставления последовательности букв языка его грамматике.

**Синтаксический анализатор (парсер)** – устройство, осуществляющее синтаксический анализ.

Синтаксический анализатор в первую очередь проверяет соответствие входной последовательности токенов (созданной ранее на этапе лексического анализа, см. лабораторную №5) грамматике языка, для которого создан парсер.

В случае соответствия входной последовательности грамматике, результатом синтаксического анализа будет некоторая структура данных, например, дерево, которое отражает структуру входной последовательности букв.

Существует много различных алгоритмов синтаксического анализа. В частности, синтаксические анализаторы можно разделить на:

* **восходящие** – сначала во входящем потоке распознаются финальные элементы дерева разбора – листья – начиная с самого левого, затем в соответствии с правилами грамматики конструируются более общие нетерминальные символы и так до корня дерева разбора – стартового нетерминала);
* **нисходящие** – процесс начинается с корня дерева – стартового символа, затем в соответствии с правилами грамматики строится предполагаемая структура дерева вниз, начиная с левой ветви, основываясь на токенах во входном потоке).

Среди восходящих парсеров можно выделить **LR(1)-парсер**, который читает входной поток и на его основании строит обратный правосторонний разбор слова. Токены из входного потока переносятся в изначально пустую память парсера, и если текущее содержимое памяти (или его часть) соответствует правой части какого-либо из правил грамматики, оно меняется на левую часть этого правила.

Цифра 1 в скобках обозначает, что при необходимости парсер может заглядывать вперед на один символ во входной последовательности, когда без этого неочевидно, какое именно правило нужно применить в данном случае.

Среди нисходящих парсеров выделяют **LL(1)-парсер**, осуществляющий левосторонний разбор для последовательности входных токенов. В данном случае, так как LL(1)-парсер – нисходящий, в начале анализа в стековой памяти парсера находится стартовый нетерминальный символ грамматики.

Далее, на основании первого токена входной последовательности выбирается правило грамматики, по которому стартовый символ меняется на последовательность терминальных и нетерминальных символов. Содержимое стека меняется так, чтобы самый левый символ правой части правила оказался на вершине стека.

Если токен входной последовательности соответствует содержимому вершины стека, то этот токен убирается и из входной последовательности, и с вершины стека. Если на вершине стека оказывается нетерминальный символ, то также на основании текущего символа входной последовательности выбирается правило, по которому этот символ в стеке меняется на последовательность других символов.

Как и LR(1), LL(1)-парсер может заглядывать на один токен входной последовательности вперед, если это требуется для корректного выбора правила грамматики.

**Задание на лабораторную работу:**

1. Для вашего любимого языка программирования составить правила грамматики, описывающей синтаксис **цикла с условием** (с предусловием или постусловием – на ваш выбор).
2. Написать программу, которая будет получать на вход строку на вашем любимом языке программирования, преобразовывать ее в последовательность лексем (см. лабораторную работу №5), проводить попытку синтаксического анализа и сообщать, соответствует ли строка грамматике цикла с условием, описанной в предыдущем пункте, или нет. Тип парсера (LL или LR можно выбрать по желанию).

**Важно!** Строить синтаксическое дерево не требуется, только определить, соответствует строка грамматике или нет.

**Важно-2!** Не стремитесь создать всеобъемлющие правила грамматики для цикла с условием, упрощения допустимы и приветствуются.

**На защите лабораторной работы необходимо** показать составленные правила грамматики и работу программы.

**В отчете надо привести:**

* Правила грамматики.
* Исходный код программы.
* Скриншоты работы программы.

**Пример правил грамматики**

Рассмотрим грамматику, описывающую простейшие арифметические операции между переменными и числами. Пусть:

**S** – стартовый нетерминальный символ.

**()+-\*/** – терминальные символы/токены входной последовательности.

**var, int, float** – также терминальные символы/токены входной последовательности, включающие в себя все переменные, натуральные и вещественные числа.

Тогда грамматика, описывающая простейшие арифметические операции между переменными и числами, может выглядеть следующим образом:

1. S → S+S

2. S → S-S

3. S → S\*S

4. S → S/S

5. S → (S)

6. S → var

7. S → int

8. S→ float

**Примеры работы алгоритма LR(1)-парсера**

Определим, соответствует ли строка **var+(var-int)/float** приведенной выше грамматике.

| **№ шага** | **Содержимое входной посл-ти** | **Содержимое памяти** | **Комментарий** |
| --- | --- | --- | --- |
| 0 | var+(var-int)/float | пусто | До начала анализа память пустая |
| 1 | +(var-int)/float | var | Первый токен переносится со входа в память (shift). |
| 2 | +(var-int)/float | S | Ищется правило, в правой части которого было бы содержимое памяти – правило №6. Осуществляется замена (reduce) одержимого памяти по соответствующему правилу. |
| 3 | (var-int)/float | S+ | Если больше нет правила, правая часть которого аналогична содержимому (или части содержимого) памяти, то выполняется операция shift. |
|  | var-int)/float | S+( | Shift |
| 4 | -int)/float | S+(var | Shift |
| 5 | -int)/float | S+(S | Reduce по правилу № 6 |
| 6 | int)/float | S+(S- | Shift |
| 7 | )/float | S+(S-int | Shift |
| 8 | )/float | S+(S-S | Reduce по правилу № 7 |
| 9 | )/float | S+(S | Reduce: S-S меняется на S по правилу №2 |
| 10 | /float | S+(S) | Shift |
| 11 | /float | S+S | Reduce: (S) меняется на S по правилу №5 |
| 12 | /float | S | Reduce по правилу №1 |
| 13 | float | S/ | Shift |
| 14 | пусто | S/float | Shift |
| 15 | пусто | S/S | Reduce: float меняется на S по правилу №8 |
| 16 | пусто | S | Reduce по правилу №1 |

Проанализирована вся входная последовательность, **в памяти находится стартовый нетерминал**, значит анализ завершился успешно, входная последовательность соответствует грамматике.

**Примеры работы алгоритма LL(1)-парсера**

Для LL-парсера несколько модифицируем грамматику:

1. S → AB

2. S → (AB)B

3. B → +S

4. B → -S

5. B → \*S

6. B → /S

7. B → ε

8 A → var

9. A → int

10. A → float

Для LL-парсеров требуется создание вспомогательной таблицы, в которой указывается, какое правило применять в зависимости от символа на вершине стека и содержимого входной последовательности. Знаком «x» в таблице помечены случаи, когда входная последовательность не соответствует грамматике (ошибка во входной последовательности).

| **Нетерминал на вершине стека** | **Текущий терминал во входной последовательности токенов:** | | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **var** | **int** | | **float** | **+** | **-** | **\*** | **/** | **(** | **)** | **EOF** |
| S | 1 | | | | x | x | x | x | 2 | x | x |
| A | 8 | 9 | 10 | | x | x | x | x | x | x | x |
| B | x | x | x | | 3 | 4 | 5 | 6 | x | 7 | 7 |

Определим, соответствует ли строка **var+(var-int)/float** приведенной выше грамматике.

| **№ шага** | **Содержимое входной посл-ти** | **Содержимое памяти** | **Комментарий** |
| --- | --- | --- | --- |
| 0 | **var**+(var-int)/float | **S** | До начала анализа в памяти содержится стартовый нетерминал |
| 1 | **var**+(var-int)/float | **A**B | В соответствии с таблицей выше меняем стартовый нетерминал по правилу 1 |
| 2 | **var**+(var-int)/float | **var**B | В соответствии с таблицей к нетерминалу A применяется правило b |
| 3 | **+**(var-int)/float | **B** | Когда самый левый токен входной последовательности совпадает с символом на вершине стека, он стирается и там, и там. В данном случае стерт был токен «var» |
| 4 | **+**(var-int)/float | **+**S | В соответствии с таблицей применяется правило 3 |
| 5 | **(**var-int)/float | **S** | Из входной последовательности и из вершины стека стирается «+» |
| 6 | **(**var-int)/float | **(**AB)B | В соответствии с таблицей применяется правило 2 |
| 7 | **var**-int)/float | **A**B)B | Стирается «(» |
| 8 | **var**-int)/float | **var**B)B | В соответствии с таблицей применяется правило 8 |
| 9 | **-**int)/float | **B**)B | Стирается «var» |
| 10 | **-**int)/float | **-**S)B | В соответствии с таблицей применяется правило 4 |
| 11 | **int**)/float | **S**)B | Стирается «-» |
| 12 | **int**)/float | **A**B)B | В соответствии с таблицей применяется правило 1 |
| 13 | **int**)/float | **int**B)B | В соответствии с таблицей применяется правило 9 |
| 14 | **)**/float | **B**)B | Стирается «int» |
| 15 | **)**/float | **)**B | В соответствии с таблицей применяется правило 7 |
| 16 | **/**float | **B** | Стирается «)» |
| 17 | /float | /S | В соответствии с таблицей применяется правило 6 |
| 18 | **float** | **S** | Стирается «/» |
| 19 | **float** | **A**B | В соответствии с таблицей применяется правило 1 |
| 20 | **float** | **float**B | В соответствии с таблицей применяется правило 10 |
| 21 | пусто | **B** | Стирается «float» |
| 22 | пусто | пусто | В соответствии с таблицей применяется правило 7 |

Проанализирована вся входная последовательность, стек пуст, значит анализ завершился успешно, входная последовательность соответствует грамматике.