Курс «Компиляторные технологии»

Лекция 2

Введение в теорию формальных языков

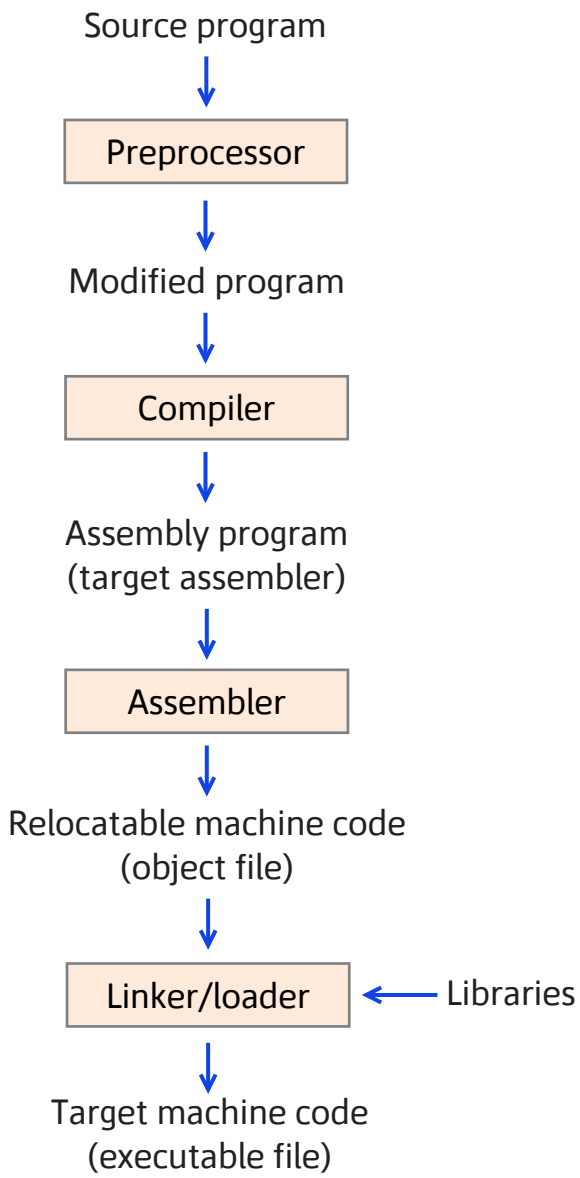
Курносов Михаил Георгиевич

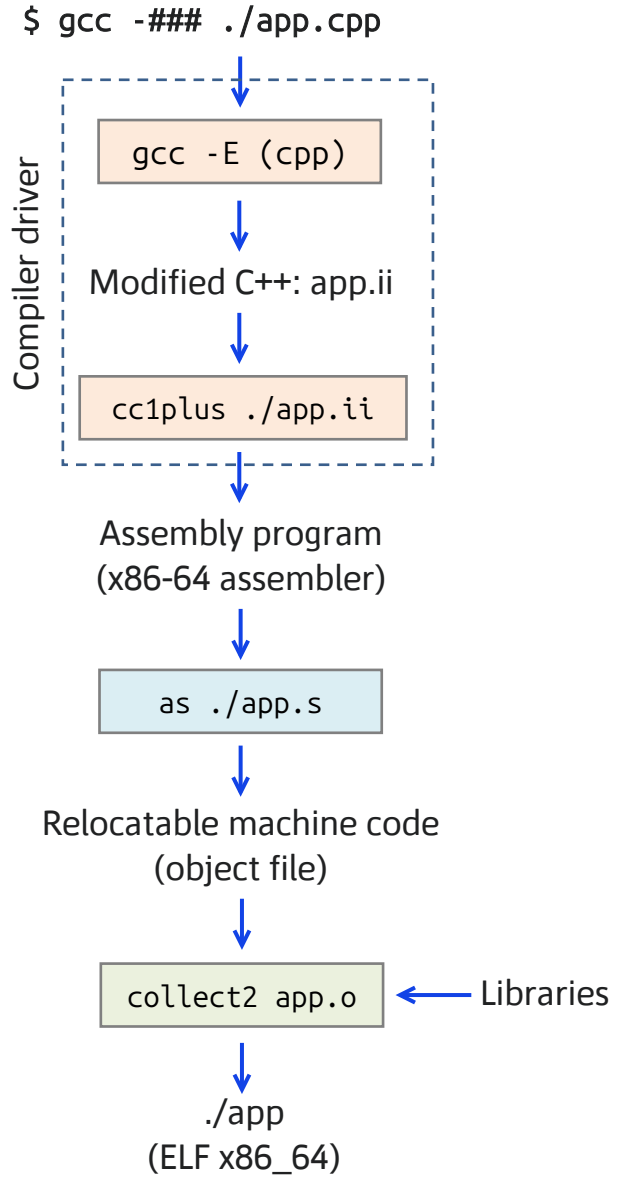
www.mkurnosov.net

Сибирский государственный университет телекоммуникаций и информатики
Осенний семестр

Процесс компиляции (повторение)

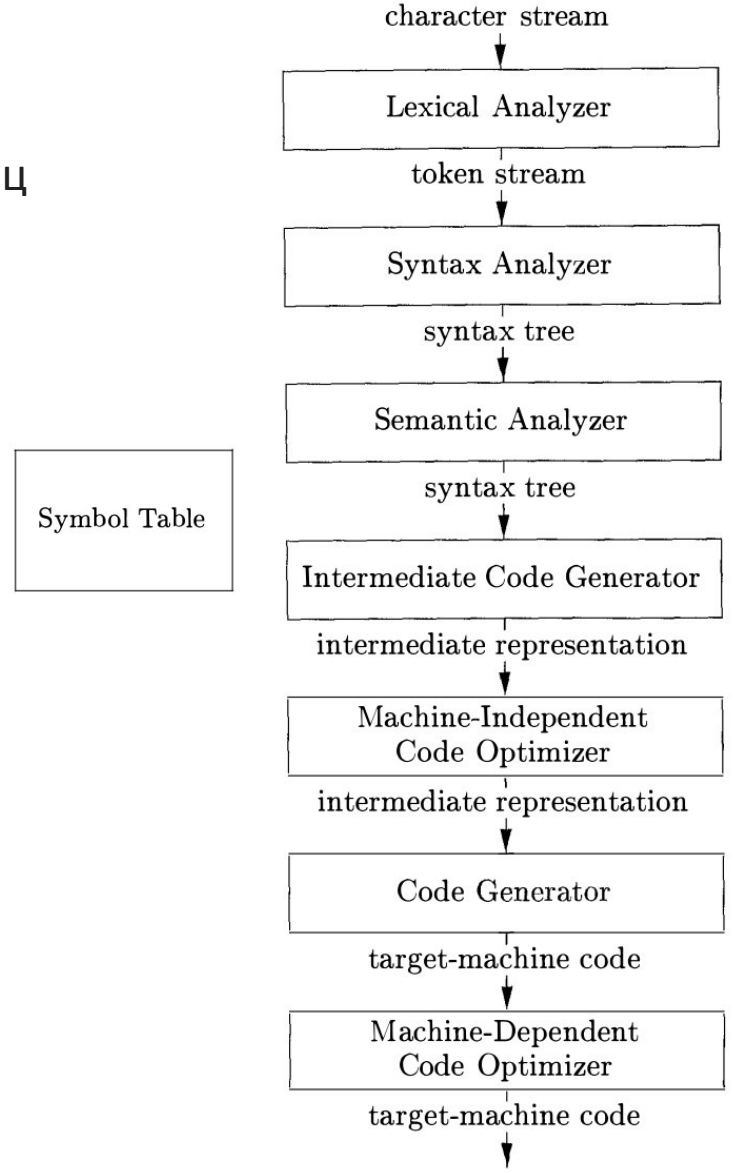
- **Препроцессор** (preprocessor) — обрабатывает директивы (#include, #define, #ifdef)
- **Компилятор** (compiler) — программа, транслирующая код на *исходном языке* (source launguage) в текст программы на *целевом языке* (target language) с сохранением семантики
- **Ассемблер** (assembler) — транслятор с ассемблера в машинный код
- **Компоновщик** (linker) — программа объединяющая объектные файлы в исполняемый файл
- **Исполняемая программа** (program) — файл на носителе информации в исполняемом формате (ELF, PE, Mach-O) с секциями кода для целевой архитектуры (target architecture)
- **Загрузчик** (loader) — загружает секции исполняемого файла в память, загружает требуемые библиотеки динамической компоновки, передает управление на точку старта





Структура компилятора (повторение)

- **Фаза анализа** (frontend, начальная стадия) —
  разбивает программу на последовательность минимально значимых единиц языка, накладывает на них грамматическую структуру языка, обнаруживает синтаксические и семантические ошибки, формирует таблицу символов, генерирует промежуточное представление программы
- **Фаза синтеза** (backend, заключительная стадия) —
  транслирует программу на основе таблицы символов и промежуточного представления в код целевой архитектуры
- **Общий процесс компиляция включает фазы** (phases):
  - лексический анализ
  - синтаксический анализ
  - семантический анализ
  - генерация (синтез) промежуточного представления
  - машинно-независимая оптимизация промежуточного представления
  - генерация машинного кода
  - машинно-зависимые оптимизации кода



[*] Aho A., Lam M., Sethi R., Ullman J. **Compilers: principles, techniques, and tools**. — 2nd ed., 2007

Содержание

- Что значить создать, задать (описать) формальный язык?
- Синтаксис языка и способы его описания
- Семантика языка и способы её описания
- Грамматика языка
- Иерархия грамматик

Язык программирования

- **Язык программирования** (programming language) — формальный язык, предназначенный для записи компьютерных программ, определяет набор лексических, синтаксических и семантических правил, определяющих внешний вид программы и действия, которые выполнит исполнитель (процессор, виртуальная машина) под её управлением

- Языки программирования:
  - императивные/декларативные
  - общего назначения/проблемно-ориентированные (domain-specific language — DSL)
  - объектно-ориентированного программирования (процедурного, структурного, ...)

- Языки разметки: HTML, XML, YAML, TeX/LaTeX

Понятие формального языка

- **Формальный язык** (formal language) — множество *L* конечных слов (строк, цепочек символов) над конечным алфавитом *A*
- **Синтаксис** (syntax) формального языка — набор правил, описывающих корректный вид его программ (цепочек символов)
- **Семантика** (semantics) формального языка — набор правил, описывающих «смысл» программ (правила области видимости переменных, совместимость типов данных, правила наследования классов, передача параметров в функции)
- Если язык состоит из конечного числа слов (корректных программ), то его можно задать перечислением множества *L*

$$L = \{a, b, hello\}, \quad A = \{a, b, h, e, l, o\}$$

- Множество корректных программ на языке C++ конечно?
- Как задать (описать) бесконечное множество цепочек символов (корректных программ)?

Формальный язык с бесконечным числом строк

- Формальный язык $L1$, состоящий из строк, содержащих произвольное целое число символов x

  Примеры цепочек (программ): x, xxx, xx, «», xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

  $L1 = \{x^n \mid n > 0\}$, где x^n — конкатенация символа x с самим собой n раз

- Язык $L2 = \{x^n y^n \mid n > 0\}$ — строки вида: один и более x, за которым следует такое же число символов y
  - *xy*
  - *xxxxxyyyyy*
  - *xxyyy — не принадлежит L2*

- Язык $L3 = \{x^m y^n \mid m, n > 0\}$ — строки вида: один и более x, за которым следует хотя бы один y
  - *xyy*
  - *xxxyy*

- Язык $L4 = \{x^m y^n \mid m, n >= 0\}$ — нуль и более x, за которым следует нуль и более y
  - *x*
  - *yy*
  - ε — пустая строка принадлежит $L4$

Регулярные выражения для описания языков

- Языки $L1$ — $L4$ имеют простую структуру (простые виды допустимых цепочек)
- Все допустимые строки языков $L1$, $L3$, $L4$ можно описать при помощи *регулярных выражений*

- Все строки языка $L4 = \{x^m y^n \mid m, n >= 0\}$ можно задать регулярным выражением $\{x^* y^*\}$
- регулярное выражение: a^* — символ a повторяется 0 или более раз (звезда Клини, Kleene star)

- Язык $L3 = \{x^m y^n \mid m, n > 0\}$, определение строк языка регулярным выражением: $\{xx^* yy^*\} = \{x^+ y^+\}$
- регулярное выражение: a^+ — символ a повторяется 1 или более раз (эквивалентно aa^*)

- **Язык записи регулярных выражений** (regular expression)
  - ab — строки из двух символов a и b (конкатенация)
  - a^* — строки из нуль или более повторений a (звезда Клини)
  - $a \mid b$ — строки из символа a или b
  - $(a \mid b)^*$ — строки из нуля или большего числа элементов a или b
    (строка из нуля или большего числа знаков, каждый из которых может быть a или b)

- Операция | имеет приоритет над *:
  - $a \mid b^*$
  - (aab | ab)*

Регулярные выражения для описания языков

- Область применения регулярных выражений — определение языковых лексем на этапе лексического анализа
- Идентификатор в языке программирования:

$$(\_|a|b|c|\ldots|z|A|B|\ldots|Z)(\_|a|b|c|\ldots|z|A|B|\ldots|Z|0|1|2|\ldots|9|)^*$$

- Целочисленный литерал в десятичной системе исчисления:

$$(1|2|\ldots|9)(0|1|2|\ldots|9)^*$$

- Ключевые слова:

$$if|for|while|do$$

Ограничения возможности регулярных выражений

- **Пример 1**. Язык $L2 = \{x^n y^n \mid n > 0\}$ — один и более x, за которым следует такое же число символов y

  xy, xxyy, xxxyyy

- Язык $L2$ невозможно описать регулярными выражениями, в регулярных выражениях не существует возможности указать, что количество элементов x должно равняться количеству элементов y
- **Пример 2**. Язык $L5$ — множество строк со сбалансированным числом скобок (Dyck language)

  (((x)))

- Существует теоретическое ограничение возможности использования регулярных выражений для описания цепочек символов для языков некоторых типов (лемма о накачке, лемма о разрастании, pumping lemma)
- **Следствие**: мы не можем описать регулярным выражением строку со сбалансированной расстановкой символов, вложенных тегов
  - HTML: <h1><h1> ... </h1></h1>
  - C++: {{{ }}}

- Требуется более общий формализм описания цепочек символов формального языка

Альтернатива регулярным выражениям — продукции

- Продукции используются для генерации множества строк языка (порождения, вывода)
- Строки языка *L*2 имеют вид:
  - $S \rightarrow xSy$
  - $S \rightarrow xy$
- Знак *продукции* (production) «→» читается как «*S* может иметь вид»
- Продукции генерируют (порождают) строки языка по следующим правилам:

  1. Начать с символа *S* и заменить его строкой, расположенной справа от знака продукции
  2. Если полученная строка не содержит символов *S*, она является строкой языка.
     В противном случае следует заменить *S* строкой после знака продукции и вернуться к п. 2

- Пример порождения:

$$\text{Язык } L2 = \{x^n y^n \mid n > 0\}$$

$$S \Rightarrow xSy \Rightarrow xxSyy \Rightarrow xxxyyy$$

- Существование цепочки порождений (вывода) говорит о принадлежности строки «xxxyyy» языку *L*2

Формальные грамматики

- **Формальная грамматика** языка (formal grammar) — способ описания формального языка, выделения подмножества L из множества всех слов некоторого конечного алфавита A
- **Порождающие** грамматики — задают правила, с помощью которых можно построить любое слово языка
- **Распознающие** (аналитические) грамматики — позволяют по данному слову определить, принадлежит оно языку или нет (синтаксически корректная программа или нет)
- **Формальная грамматика** G — это описание формального языка (его синтаксиса) четверкой

$$\boldsymbol{G = (V_T, V_N, P, S),}$$

где

- V_T — алфавит, символы которого называют терминальными символами (терминалами, terminal);
- V_N — алфавит с нетерминальными символами (нетерминалами, nonterminal);
- P — множество продукций (правил), каждый элемент которого состоит из пары (a, b), где a — левая часть продукции, b — правая часть продукции, а продукция записывается: $a \to b$;
- S — начальный символ грамматики (start symbol)

$$V = V_T \cup V_N, \quad V_T \cap V_N = \varnothing$$

Формальные грамматики

- **Грамматика** используется для генерации последовательностей символов, составляющих строки языка, начиная со стартового символва S и последовательно заменяя его или нетерминалы, которые появятся позднее, с помощью одного из порождений грамматики
- На каждом этапе к нетерминалу из левой части применяется продукция, заменяющая этот нетерминал последовательностью символов своей правой части
- Процесс прекращается после получения строки, состоящей только из терминальных символов (не содержащей нетерминалов)
- Языку *принадлежат те, и только те строки символов, которые можно получить с помощью заданной грамматики* (породить, вывести)
- **Пример**: грамматика языка $L2 = \{x^n y^n \mid n > 0\}$

$$G = (V_T, V_N, P, S),$$

- $V_T = \{x, y\}$
- $V_N = \{S\}$
- $P = \{$
  $S \rightarrow xSy,$
  $S \rightarrow xy$

}

Формальные грамматики

- **Язык** $L4 = \{x^m y^n \mid m, n >= 0\}$ — нуль и более x, за которым следует нуль и более y
  - x
  - yy
- ε — пустая строка принадлежит $L4$
- **Грамматика** для языка $L4$:

$$G4 = (V_T, V_N, P, S),$$

- $V_T = \{x, y\}$
- $V_N = \{S, B\}$
- $P = \{$

  $S \to xS,$

  $S \to yB,$

  $S \to x,$

  $S \to y,$

  $B \to yB,$

  $B \to y,$

  $S \to \varepsilon$

$\}$

Пример вывода строки «xxyyy» из грамматики:

$$S \Rightarrow xS \Rightarrow xxS \Rightarrow xxyB \Rightarrow xxyyB \Rightarrow xxyyy$$

1. S ⇒ xS — порождение из первой продукции
2. S ⇒ xS ⇒ xxS — порождение из первой продукции)
3. S ⇒ xS ⇒ xxS ⇒ xxyB — порождение из второй продукции
4. S ⇒ xS ⇒ xxS ⇒ xxyB ⇒ xxyyB — порождение из пятой продукции
5. S ⇒ xS ⇒ xxS ⇒ xxyB ⇒ xxyyB ⇒ xxyyy — порождение из шестой продукции

Эквивалентные грамматики

- Для генерации языка обычно не существует единственной грамматики
- **Язык** $L4 = \{x^m y^n \mid m, n >= 0\}$ — нуль и более x, за которым следует нуль и более y

Грамматика $G1$ = (V_T, V_N, P, S),

- $V_T = \{x, y\}$
- $V_N = \{S, B\}$
- P = {
  $S \to xS$,
  $S \to yB$,
  $S \to x$,
  $S \to y$,
  $B \to yB$,
  $B \to y$,
  $S \to \varepsilon$
  }

Грамматика $G2$ = (V_T, V_N, P, S),

- $V_T = \{x, y\}$
- $V_N = \{X, Y\}$
- P = {
  $S \to XY$,
  $X \to xX$,
  $X \to \varepsilon$,
  $Y \to yY$,
  $Y \to \varepsilon$
  }

Пример вывода строки «xxyyy» из $G2$:
$S \Rightarrow XY \Rightarrow xXY \Rightarrow xxXY \Rightarrow xxY \Rightarrow xxyY \Rightarrow xxyyY \Rightarrow xxyyy$

- Грамматики $G1$ и $G2$ генерируют язык $L4$
- Две грамматики, генерирующие один и тот же язык, называются ***эквивалентными***

Формальные грамматики общего вида

- **Формальная грамматика** G — это описание формального языка (его синтаксиса) четверкой

$$G = (V_T, V_N, P, S),$$

где P — множество продукций, продукции: $a \to b$

- В общем случае левые части продукций могут содержать более одного символа

- Пример грамматики $G = (\{a\}, \{S, N, Q, R\}, P, S)$:

P = {
  $S \to QNQ$,
  $QN \to QR$, // N можно заменить на R, только если N следует после Q
  $RN \to NNR$,
  $RQ \to NNQ$, // R можно заменить на NN, только если после R следует Q
  $N \to a$,
  $Q \to \varepsilon$
}

- Продукции 2, 4 являются ***контекстно-зависимыми*** (context-sensitive production)

Формальные грамматики общего вида

- Грамматика $G = (\{a\}, \{S, N, Q, R\}, P, S)$:

  P = {
  $S \to QNQ$,
  $QN \to QR$, // N можно заменить на R, только если N следует после Q
  $RN \to NNR$,
  $RQ \to NNQ$, // R можно заменить на NN, только если после R следует Q
  $N \to a$,
  $Q \to \varepsilon$
  }

- Порождает язык $\{a^m \mid m = 2^k, k = 1, 2, ...\} = \{aa, aaaa, aaaaaaaa, ...\}$, m — положительная степень двойки

- $S \Rightarrow$ QNQ $\Rightarrow$ QRQ $\Rightarrow$ QNNQ $\Rightarrow$ ... $\Rightarrow$ aa

- S $\Rightarrow$ QNQ $\Rightarrow$ QRQ $\Rightarrow$ QNNQ $\Rightarrow$ *QRNQ* $\Rightarrow$ QNNRQ $\Rightarrow$ QNNNNQ $\Rightarrow$ ... $\Rightarrow$ *aaaa*

Иерархия Хомского

- **Иерархия Хомского** — классификация формальных языков и формальных грамматик на 4 типа по их условной сложности [Ноам Хомский, 1956, https://chomsky.info/wp-content/uploads/195609-.pdf]
- Для отнесения грамматики к определенному типу необходимо соответствие всех её продукций некоторым схемам

Тип	Грамматика	Вид продукций	Применение
Тип 0	**Неограниченные грамматики** (рекурсивно перечислимые, recursively enumerable)	$a \to b$ ▪ a $\in V^+$ — непустая цепочка, содержащая хотя бы один нетерминал ▪ $b \in V^*$ — любая цепочка символов из $V = V_T \cup V_N$ (эквивалентны машинам Тьюринга)	Практического применения в силу своей сложности (общности) такие грамматики не имеют
Тип 1	**Контекстно-зависимые** (context-sensitive)	$aAb \to acb$ ▪ a, $b \in V^*$ — любая цепочка символов из V ▪ $c \in V^+$ — непустая цепочка из V ▪ $A \in V_N$ (эквивалентны линейно ограниченным автоматам)	Анализ текстов на естественных языках, при построении компиляторов практически не используются
Тип 2	**Контекстно-свободные** (context-free)	$A \to b$ ▪ $A \in V_N$ ▪ $b \in V^*$ — любая цепочка символов из V (эквивалентны магазинным автоматам)	Описание синтаксиса компьютерных языков
Тип 3	**Регулярные** (regular)	$A \to Bc$ или $A \to c$ — леволинейные грамматики $A \to cB$ или $A \to c$ — праволинейные грамматики ▪ $A, B \in V_N$ ▪ $c \in V_T^*$ (эквивалентны конечным автоматам)	Описание простейших конструкций: идентификаторов, строк, констант, языков ассемблера, командных процессоров

Иерархия Хомского

- **Иерархия Хомского** — классификация формальных языков и формальных грамматик на 4 типа по их условной сложности [Ноам Хомский, 1956]
- Для отнесения грамматики к определенному типу необходимо соответствие всех её продукций некоторым схемам

Тип	Грамматика	Вид продукций	Применение
Тип 0	**Неограниченные грамматики** (рекурсивно перечислимые, recursively enumerable)	$a \to b$ ▪ a ∈ V^+ — непустая цепочка, содержащая хотя бы один нетерминал ▪ $b \in V^*$ — любая цепочка символов из	Практического применения в силу своей сложности (общности) такие грамматики не имеют
Тип 1			
Тип 2			
Тип 3		(эквивалентны конечным автоматам)	

- Один и тот же язык может быть задан разными грамматиками, относящимися к разным типам
- Язык относится к наиболее простому из типов грамматик, которыми может быть описан
- Например, формальный язык, описанный грамматикой с фразовой структурой, контекстно-зависимой и контекстно-свободной грамматиками, будет контекстно-свободным
- Наиболее сложные — языки с фразовой структурой (сюда можно отнести естественные языки), далее — КЗ-языки, КС-языки и самые простые — регулярные языки

Компьютерное описание контекстно-свободных грамматик

- **Форма Бэкуса-Наура** (Backus–Naur Form — BNF) // Algol-58-60

```
<expression> ::= <term> | <term> "+" <expression>
<term>       ::= <factor> | <factor> "*" <term>
<factor>     ::= <constant> | <variable> | "(" <expression> ")"
<variable>   ::= "x" | "y" | "z"
<constant>   ::= <digit> | <digit> <constant>
<digit>      ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

Поддержка БНФ

- ANTLR
- Coco/R
- JavaCC
- Bison
- Yacc

- **Расширенная форма Бэкуса-Наура** (Extended Backus–Naur Form — EBNF) // N. Wirth, 1977
  ISO/IEC 14977 (1996). Синтаксический метаязык – Расширенная Форма Бэкуса-Наура

```
expression = term , [ "+" , expression ];
term       = factor , [ "*" , term ];
factor     = constant | variable | "(" , expression , ")";
variable   = "x" | "y" | "z";
constant   = digit , { digit };
digit      = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
```

- **Pascal, 1973 //** https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/68910/eth-3059-01.pdf

Компьютерное описание контекстно-свободных грамматик

- **Синтаксические диаграммы** (syntax diagram) — графическое представление БНФ