Введение в синтаксический анализ

среда, 1 декабря 2021 г. 16:43

Формальная грамматика - способ описания формального языка, т.е. выделение некоторого подмножества из множества всех слов некоторого конечного алфавита

Иерархия Хомского

- Формальные языки
  - Неограниченные (грамматики с фразовой структурой, т.е. все без исключения формальный грамматики вида $\alpha \to \beta$)
  - Контекстно-зависимые (частный случай формальной грамматики, у которой левые и правые части всех продукций могут быть окружены терминальными и нетерминальными символами)
  - Контекстно-свободные (частный случай формальной грамматики, у которой левые части всех продукций являются одиночными нетерминалами и не имеющими конкретного символьного значения)
  - Регулярные

Регулярные языки

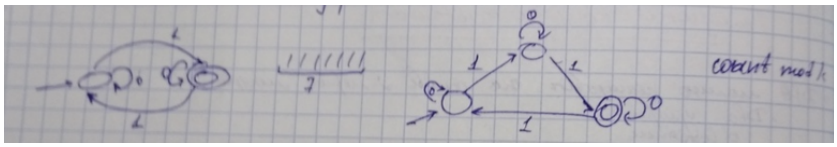
- Самые ограниченные формальные языки
- Много применений

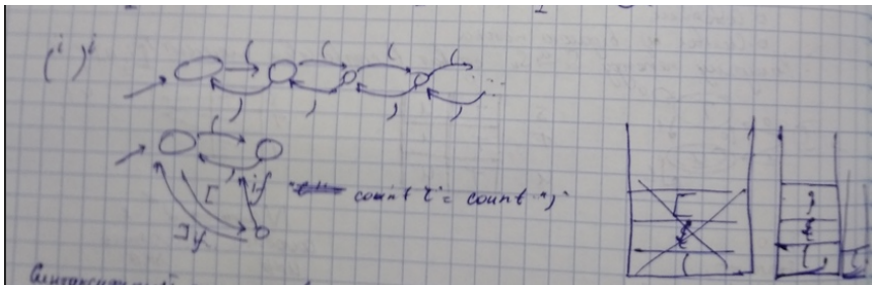
$$\left\{ (^i)^i \mid i \geq 0 \right\}$$

```
()
(())
((()))
...
```

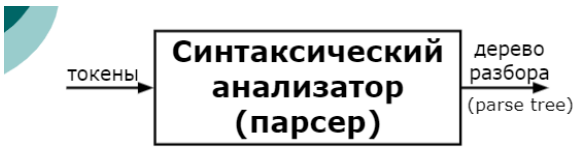
```
((1 + 2) * 3)
```

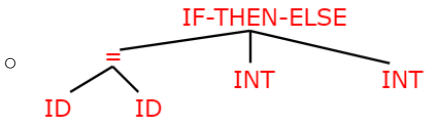
```
if ... then
  if ... then
  fi;
fi;
```





Синтаксический анализатор (парсер)



- Язык
  - If x = y then 1 else 2 fi
- Входные данные для парсера
  - IF ID = ID THEN INT ELSE INT FI
- Результат работы парсера
  - 

Структура компилятора

Фаза компиляции	Входные данные	Выходные данные
Лексический анализ	Строки символов	Токены
Синтаксический анализ	Токены	Дерево разбора (может быть неявным)

Контекстно-свободные грамматики

- Не все последовательности токенов являются программами…
- Парсер должен различать корректные и некорректные последовательности токенов
- Нужны
  - Язык описания корректных последовательностей токенов
  - Метод проверки корректности последовательностей токенов
- Языки программирования имеют рекурсивную структуру
- Например,
  - STATEMENT - это if EXPR then STATEMENT else STATEMENT
  - While EXPR do STATEMENT
  - Repeat STATEMENT until EXPR
- Контекстно-свободные грамматики-удобный способ задания таких рекурсивных структур

- Контекстно-свободная грамматика
  - Множество терминальных символов T
  - Множество нетерминальных символов N
  - Начальный символ S, $S \in N$
  - Множество продукций

$$X \to Y_1 \square Y_n$$

- $X \in N$

$$Y_i \in T \cup N \cup \{\varepsilon\}$$

- $S \to (S) \qquad N = \{S\}$

$$S \to \varepsilon \qquad T = \{(,)\}$$

  - Продукции можно рассматривать как правила подстановки S --> (S)

Контекстно-свободные грамматики

1. Начать со строки, состоящей из единственного стартового символа S
2. Заменить любой нетерминальный символ X на правую часть какой-либо из его продукций X --> $Y_1 \ldots Y_n$
3. Повторять шаг 2, пока остаются нетерминальные символы

$$X_1 \square X_i X X_{i+1} X_n \to$$

Продукция: $X \to Y_1 \square Y_n$

$$S \to \_\_ \to \_\_ \to \alpha_0 \to \alpha_1 \to \alpha_2 \to \square \to \alpha_n$$

$\alpha_0 \xrightarrow{*} \alpha_n$ (за 0 или более шагов)

Пусть G — контекстно-свободная грамматика с начальным символом S.
Тогда язык L(G) — это:

$$\{a_1 \square\ a_n \mid \forall i \quad a_i \in T \cap S \xrightarrow{*} a_1 \square\ a_n\}$$

- Терминальные символы - символы, для которых в грамматике нет правил подстановки
- Появившись в строке, терминалы больше не могут быть ничем заменены
- Терминалы - токены языка

EXPR --> if EXPR then EXPR else EXPR fi
EXPR --> while EXPR loop EXPR pool
EXPR --> id
…

Строки, которые можно получить

- id
  if id then id else id fi
  while id loop id pool
  if while id loop id pool then id else id fi
  if if id then id else id fi then id else id fi
- Простые арифметические выражения

```
E  →  E + E          id
   |  E * E          id + id
   |  (E)            id + id * id
   |  id             (id + id) * id
                     ...
```

- Идея контекстно-свободных грамматик - важный шаг. Но
  - Принадлежность строки определяется в виде да/нет; еще необходимо строить дерево разбора
  - Необходимо обрабатывать ошибки во входных данных (программах)
  - Нужна реализация контекстно-свободных грамматик
- Способ записи грамматики имеет значение
  - Многие грамматики порождают один и тот же язык
  - Инструменты чувствительны к способам записи грамматики

Порождение (цепочки вывода)

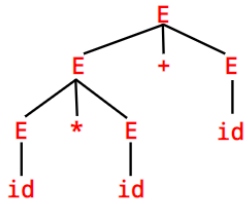
Порождение - последовательность продукций
S --> … --> … --> …

- Порождение может быть представлено деревом
  - Стартовый символ - корень дерева
  - Для продукции X --> $Y_1 \ldots Y_n$ к узлу X добавляются дочерние узлы $Y_1 \ldots Y_n$

- Грамматика:
  `E → E+E | E*E | (E) | id`
  Строка:
  `id * id + id`

Порождение	Дерево разбора (синтаксическое дерево)

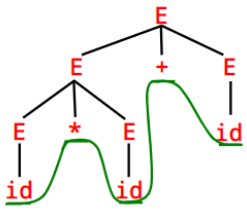
- E
  → E + E
  → E * E + E
  → id * E + E
  → id * id + E
  → id * id +
  id



Дерево разбора

- Синтаксическое дерево состоит из
  - Терминалов - листьев
  - Нетерминалов - промежуточных узлов
- Последовательный обход листьев позволяет восстановить исходную строку
- Показывает ассоциативность операторов (в отличие от исходной строки)
- Пример - левое порождение (left-most derivation)
  - На каждом шаге заменяется самый **левый** нетерминал
- Существует также правое порождение, где на каждом шаге производится замена самого правого нетерминала
- Левому и правому порождению соответствует одно и тоже дерево разбора
- Интерес представляет не только вопрос: верно ли, что S ∈ L(G)
  - Необходимо построить синтаксическое дерево для S
- Порождение определяет дерево разбора
  - Но одно и то же дерево разбора может иметь много порождений
- Левые и правые порождения играют важную роль в синтаксическом анализе

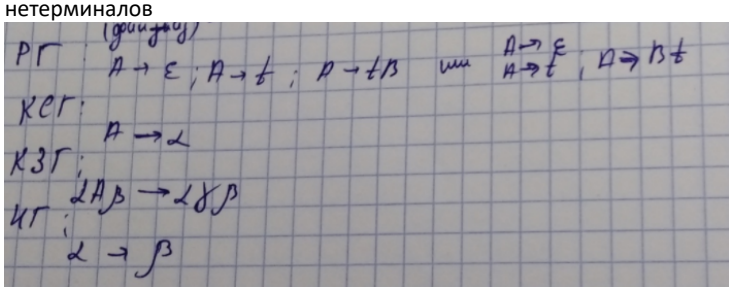
E
→ E + E
→ E * E + E
→ id * E + E
→ id * id + E
→ id * id +
id



Формальный язык - множество строк, удовлетворяющих некоторой грамматике
Грамматика - набор правил подстановки (продукций), задающих некоторый формальный язык
Терминал - символ в грамматике, для которого отсутствует правила подстановки. Для ЯП обычно - лексемы
Нетерминал - символы в грамматике, для которых есть правила подстановки. В ЯП обычно - какой-либо синтаксической конструкции: выражение, оператор, объявление переменной, диапазонов, и так далее
Начальный стартовый символ - весь текст программы/модуля

Обозначения записи грамматики

Заглавные латинские буквы - нетерминалы
Строчные латинские буквы - терминалы
Греческие буквы - произвольные последовательности терминалов и нетерминалов



Порождения

- Интерес представляет не только ответ на вопрос, верно ли что $S \in L(G)$
  - Необходимо построить синтаксическое дерево для S
- Порождение определяет дерево разбора
  - Но одно и то же дерево разбора может составляться разными порождениями