Методы трансляции

Михайлов Максим

4 сентября 2021 г.

Оглавление

Лекция 1 2 сентября **2**
1 Введение 2
2 Нисходящие парсеры 3

Лекция 1

2 сентября

1 Введение

Этот курс — про парсеры. Рассмотрим их работу в общем случае.

1. На вход подается строка.
2. Строка разбивается на неделимые блоки *(лексемы или токены)* лексическим анализом.
3. Последовательность токенов с учетом синтаксиса языка переводится в дерево разбора путем синтаксического разбора *(парсинга)*.
4. Дерево разбора не есть самоцель, дерево переводится с учетом семантики языка в искомый результат.

Адепты architecture-driven подхода могут захотеть разделить семантику и синтаксис, однако это проблематично. Рассмотрим арифметические выражения как пример.

Токены арифметических выражений это $+, \cdot, (,), n$, где n — число. Синтаксис задается следующей контекстно-свободной грамматикой:

- $E \to n$
- $E \to (E)$
- $E \to E + E$
- $E \to E \cdot E$

Однако, эта грамматика не однозначна, и выражение $2 + 2 \cdot 2$ можно разобрать поразному, из-за чего невозможно навесить семантику. Таким образом, синтаксис нужно задавать с учетом семантики:

- $E \to T$

- $E \to T + E$
- $T \to F$
- $T \to F \cdot T$
- $F \to n$
- $F \to (E)$

Но с такой грамматикой операции правоассоциативные и семантику не получится навесить с добавлением вычитания. В правильной грамматике нужно переставить местами аргументы второго правила.

Рассмотрим, как мы будем писать калькулятор арифметических выражений по дереву разбора. Наивный подход — обойти дерево DFS-ом и рассматривать детей вершины, в которой мы находимся. Однако, таким образом информация о синтаксисе описывается в двух сущностях — в парсере и в калькуляторе. Это неудобно, поэтому часто парсинг и вычисления комбинируются в один шаг без построения дерева разбора. На примере арифметических выражений:

- $E_0.val = T.val$
- $E_0.val = E_1.val + T.val$
- $E_0.val = E_1.val - T.val$
- $\vdots$

Такой подход называется **синтаксически управляемая трансляция**.

Итого существуют четыре подхода дизайну систем парсинга в зависимости от сложности задачи:

1. Ad hoc: без теории, наивно.
2. Parser + walker: Парсер производит дерево разбора и walker его обходит.
3. Синтаксически управляемая трансляция.
4. Декомпозиция задач.

Этот курс рассматривает второй и третий подходы.

2 Нисходящие парсеры

Рассмотрим пример калькулятора арифметических выражений:

```
int expr():
    r = term()
    nexttoken()
```

```
    while token == '+':
        nexttoken()
        t = term()
        r += t

int term():
    r = factor()
    nexttoken()
    while token == '*':
        nexttoken()
        f = factor()
        r += f

int factor():
    if token == '('
        nexttoken()
        r = expr()
        assert token == ')'
        nexttoken()
    else # token = 'n'
        r = tokenval()
        nexttoken()
```

Какая связь между этим кодом и грамматикой арифметических выражений? Оказывается, весьма близкая и код можно получить из нее.

Определение (контекстно-свободная грамматика).

- Алфавит Σ — множество токенов
- Нетерминалы N
- Стартовый нетерминал $S \in N$
- Правила $P \subset N \times (N \cup \sum)^*$

Определение. $\langle A, \alpha \rangle \in P \Leftrightarrow A \to \alpha$

Определение. $\alpha \Rightarrow \beta$ — из α выводится за один шаг β, если:

- $\alpha = \alpha_1 A \alpha_2$
- $\beta = \alpha_1 \xi \alpha_2$
- $A \to \xi \in P$

Определение (язык грамматики). $L(\Gamma) = \{x \mid S \Rightarrow^* x\}, x \in \Sigma^*$, где $\Rightarrow^*$ есть замыкание отношения $\Rightarrow$.

Определение. Грамматика **однозначна**, если для любого слова из языка есть только одно дерево разбора и **неоднозначна** иначе.

Примечание. Здесь и далее буквы из конца латинского алфавита обозначают нетерминалы, а буквы греческого алфавита — строки из терминалов и/или нетерминалов.

Определение. $\Gamma \in LL(1)$, если из выполнения следующих двух условий:

- $S \Rightarrow^* xA\alpha \Rightarrow x\xi\alpha \Rightarrow^* xcy$
- $S \Rightarrow^* xA\beta \Rightarrow x\eta\beta \Rightarrow^* xcz$

следует $c \in \Sigma$, или $c = \varepsilon$, или $y = \varepsilon$, или $z = \varepsilon$, тогда $\xi = \eta$.

Определение. $\Gamma \in LL(k)$, если из выполнения следующих двух условий:

- $S \Rightarrow^* xA\alpha \Rightarrow x\xi\alpha \Rightarrow^* xcy$
- $S \Rightarrow^* xA\beta \Rightarrow x\eta\beta \Rightarrow^* xcz$

следует $c \in \Sigma^k$, или $c \in \Sigma^{\leq k}$, или $y = \varepsilon$, или $z = \varepsilon$, тогда $\xi = \eta$.

В частности, $LL(0)$ — линейные программы.

$LL(1)$ грамматики есть класс всех грамматик, которые можно разобрать рекурсивным спуском.

Определение $LL(1)$ грамматик не конструктивно, т.к. проверка определения может длиться бесконечно *(по количеству всех выводов)*. Определим конструктивный критерий принадлежности $LL(1)$, для этого мы рассмотрим две вспомогательные функции:

- FIRST: $(N \cup \Sigma)^* \to 2^{\Sigma \cup \{\varepsilon\}}$
- FOLLOW: $N \to 2^{\Sigma \cup \{\$\}}$

$$\text{FIRST}(\alpha) \coloneqq \{c \mid \alpha \Rightarrow^* c\beta\} \cup \{\varepsilon \mid \alpha \Rightarrow^* \varepsilon\}$$
$$\text{FOLLOW}(A) \coloneqq \{c \mid S \Rightarrow^* \alpha A c\beta\} \cup \{\$ \mid S \Rightarrow^* \alpha A\}$$

Примечание. Мы считаем, что в грамматике нет нетерминалов, из которых нельзя вывести строку из терминалов. Это допущение не теряет общности, т.к. существует алгоритм удаления “бесполезных” нетерминалов, см. курс дискретной математики.

Теорема 1. $\Gamma \in LL(1) \Leftrightarrow A \to \alpha, A \to \beta$:

1. $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \varnothing$
2. $\varepsilon \in \text{FIRST}(\alpha) \Rightarrow \text{FIRST}(\beta) \cap \text{FOLLOW}(A) = \varnothing$