

Нисходящий анализ

Левая рекурсия

Грамматика называется **непосредственно леворекурсивной**, если $\exists(A \rightarrow A\alpha) \in P$. Плохо, потому что грамматика не разделённая. Просто **леворекурсивной**, если $\exists(A \Rightarrow^+ A\alpha)$.

Наталкиваясь на такую штуку, мы не можем посчитать, сколько раз было применено такое правило. Потенциальный бесконечный вывод. Но! Существует алгоритм, делающий леворекурсивную грамматику нормальной.

$A \rightarrow A\alpha_1 | \dots | A\alpha_n | \beta_1 | \dots | \beta_m \quad \forall i : \beta_i \text{ не начинается с } A$

A' — новый нетерминал

Заменяем все рекурсивные правила на такую группу

$A \rightarrow \beta_1 A' | \dots | \beta_m A'$

$A' \rightarrow \alpha_1 A' | \dots | \alpha_n A' | \lambda$

Левая рекурсия - это накопление альфы, и добавление в конце беты. Давайте сначала поставим бету, а потом накопим альфы, перейдя к правой рекурсии.

Но так можно устранить только непосредственную рекурсию! Нужен алгоритм для общего случая.

Алгоритм устранения левой рекурсии

Недостаток — на вход нужно подавать λ -свободную грамматику. И ацикличную, по Dragon book

Ввод: λ -свободная грамматика.

Суть: находим правила, где правая часть начинается с предыдущего нетерминала. Заменяем его на то, что из него выводится.

$\Gamma = \{A_1 \dots A_n\}$ — упорядочим все нетерминалы

for $i = 1 \dots n$:

for $j = 1 \dots i - 1$: $\#j < i$

 Все правила вида $A_i \rightarrow A_j\alpha$ заменить на $A_i \rightarrow \beta\alpha$, где $(A_j \rightarrow \beta) \in P$

 Устранить непосредственную рекурсию для A_i

Доказательство корректности — индукция по i

$(A_i \rightarrow A_j\alpha) \in P \Rightarrow i < j$

БИ $i = 1$ — пропустили внутренний цикл, если рекурсия и была, то она была непосредственной: $A_1 \rightarrow A_1\alpha$. Значит, во всех остальных продукциях вида $A_1 \rightarrow A_k\alpha$ — $k > 1$

ПИ После $i - 1$ -ой итерации внешнего цикла все нетерминалы A_m , где $m < i$ стали "чистыми" — в продукциях вида $A_m \rightarrow A_k\alpha$ — $k > m$

То есть рекурсия если и есть, то только в необработанных правилах

ШИ $A_i \rightarrow A_m\alpha$, где $m < i$

Начинаем выполнять внутренний цикл. Пусть $A_m \rightarrow \beta\gamma \in P$.

Если β начинается с нетерминала A_k , то $k > m$ по ПИ, так как $m < i$. То есть после внутреннего цикла — $m \geq i$. Равенство — непосредственная рекурсия. Устранили, получили строгое неравенство.

■

Пример

$$S \rightarrow Aa|AB|B$$

$$A \rightarrow SB|ac$$

$$B \rightarrow Ac|b$$

Надо перенумеровать:

$$S_1 \rightarrow A_2a|A_2B_3|B_3$$

$$A_2 \rightarrow S_1B_3|ac \text{ [зачёркнуто после 2 итерации]}$$

$$B_3 \rightarrow A_2c|b$$

После внутреннего цикла:

$$A_2 \rightarrow A_2aB_3|A_2B_3B_3|B_3B_3|ac \text{ [зачёркнуто после 2 итерации]}$$

Добавим A' :

$$A_2 \rightarrow B_3B_3A'|acA'$$

$$A' \rightarrow aB_3A'|B_3B_3A'|\lambda$$

Третья итерация:

$$B_3 \rightarrow B_3B_3A'c|acA'c|b$$

Устраним непосредственную рекурсию:

$$B_3 \rightarrow acA'cB'|bB'$$

$$B' \rightarrow B_3A'cB'|\lambda$$

Готово.

Левая факторизация

$$A \rightarrow \beta\alpha_1|\beta\alpha_2|\dots$$

Факторизация — устранение всех общих префиксов.

Альтернатива — все правые части одного нетерминала.

Почему плохо для нисходящего анализа? Из бет выводится одно и то же, и нам нужно пройти на неопределённую глубину, чтобы понять, какое правило было применено.

■ $S \rightarrow \text{if}(B)S|\text{if}(B)S \text{ else } S$ — не сможем узнать, а был ли else

Алгоритм

Для каждого нетерминала найдём самый длинный общий префикс среди его альтернатив. Необязательно задействовать все альтернативы, можно хотя бы две. Затем введём новый нетерминал A' и заменим исходные правила $A \rightarrow \beta\alpha_1|\beta\alpha_2$ на:

- $A \rightarrow \beta A'$
- $A' \rightarrow \alpha_1 | \alpha_2$

Продолжаем, пока у альтернатив есть общий префикс.

Доказательство корректности

Множество выводимых из A цепочек никак не меняется, просто вместо $A \Rightarrow \beta \alpha$ получаем $A \Rightarrow \beta A' \Rightarrow \beta \alpha$. Другие нетерминалы вообще не трогаем, так что и с ними всё хорошо.

У новых правил не будет общего префикса с исходными правилами, потому что префикс выбирается максимальный из всех. И рано или поздно все общие префиксы исчезнут.

Пример

$S \rightarrow Abc | AbB | AC' | ABB$ [зачёркнуто после первой итерации]

$A \rightarrow Bc | b$

$B \rightarrow aa$

$C \rightarrow aA$

Самый длинный общий префикс — Ab

$S \rightarrow AbD | AC' | ABB$ [зачёркнуто после второй итерации]

$D \rightarrow c | B$

Самый длинный общий префикс — A

$S \rightarrow AE$

$E \rightarrow bD | C' | BB$

Готово.

$LL(1)$ -грамматики

[небольшая презентация.pdf](#)

[Понятный ответ](#) на тему "Зачем вообще нужны эти FIRST и FOLLOW и как они связаны с SELECT"

Чего мы хотим? В момент обозревания на стеке какого-то нетерминала и какого-то символа на входе, знать, какую команду нужно применять.

$(B, a) \rightarrow (j, _)?$

$(B \rightarrow \gamma) \in P$

$u | v$

B

$\beta \quad uB\beta$ — левая форма

∇

Пусть $v = av'$. Тогда либо из B должно выводиться что-то, начинающееся с a , либо, если B аннулируется, тогда из β должно выводиться что-то, начинающееся с a .

FIRST

Опр. $FIRST(\alpha) \subseteq \Sigma \cup \{\lambda\}$:

$$a \in FIRST(\alpha) \iff \alpha \Rightarrow^* a\alpha'$$

$$\lambda \in FIRST(\alpha) \iff \alpha \Rightarrow^* \lambda$$

$FIRST(\alpha)$ — все терминалы, с которых могут начинаться всевозможные выводы из α .

Нисходящий анализ умеет работать с аннулирующими правилами. А с левой рекурсией нет. Поэтому ничего страшного, если в процессе избавления от левой рекурсии появляются аннулирующие правила.

Пример

$$S \rightarrow AC$$

$$A \rightarrow abC|bB$$

$$B \rightarrow b$$

$$C \rightarrow c|\lambda$$

$$FIRST(AC) = \{a, b\}$$

$$FIRST(CA) = \{c, a, b\}$$

FOLLOW

Опр. $FOLLOW(A) \subseteq \Sigma \cup \{\vdash\}$:

$$a \in FOLLOW(A) \iff S \Rightarrow^* \alpha A a \beta$$

$$\vdash \in FOLLOW(A) \iff S \Rightarrow^* \alpha A$$

$$FOLLOW(A) = \{c, \vdash\}$$

Множество терминалов, которые могут встретиться непосредственно справа от нетерминала A в некоторой цепочке.

SELECT

Опр. $SELECT(A \rightarrow \alpha)$:

1. $FIRST(\alpha)$, если $\lambda \notin FIRST(\alpha)$
2. $FIRST(\alpha) \setminus \{\lambda\} \cup FOLLOW(A)$, иначе

Множество выбора правил.

$$SELECT(A \rightarrow \alpha) = FIRST(\alpha FOLLOW(A))$$

$LL(1)$ -грамматика

Опр. $LL(1)$ -грамматика:

$$\forall A \in \Gamma : \forall (A \rightarrow \beta), (A \rightarrow \alpha) \in P:$$

$$SELECT(A \rightarrow \alpha) \cap SELECT(A \rightarrow \beta) = \emptyset$$

Множество $SELECT(A \rightarrow \alpha)$ хранит в себе множество символов, увидя который, нужно применить правило $A \rightarrow \alpha$. Если вдруг эти множества пересекаются, то мы не можем однозначно выбрать правило.

LL — две левых стороны. Читаем слева направо, восстанавливаем левый вывод.

1 — достаточно прочитать один символ со входа, чтобы понять, что делать дальше

Пример

$$E \rightarrow E + T | T$$

$$T \rightarrow T * F | F$$

$$F \rightarrow (E) | x$$

Устраним рекурсию

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' | \lambda$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' | \lambda$$

$$F \rightarrow (E) | x$$

	FOLLOW
E	$\neg,)$
E'	$\neg,)$
T	$+, \neg,)$
T'	$\neg, +,)$
F	$\neg, +, *,)$

	FIRST
TE'	$(, x$
$+TE'$	$+$
λ	λ
FT'	$(, x$
$*FT'$	$*$
(E)	$($
x	λ

Предложение. Если грамматика G содержит леворекурсивное правило, то G — не LL(1)

$$(A \rightarrow A\alpha) \in P$$

$$(A \rightarrow \beta) \in P, \text{ где } \beta[1] \neq A$$

$$SELECT(A \rightarrow A\alpha) \cap SELECT(\beta) \neq \emptyset$$

$$1. a \in FIRST(\beta) \Rightarrow a \in FIRST(A) \subseteq FIRST(A\alpha)$$

$$! FIRST(A) = \bigcup_{(A \rightarrow \beta) \in P} FIRST(\beta)$$

$$2. FIRST(\beta) = \{\lambda\} \text{ — из } \beta \text{ выводится только } \lambda$$

$$1. a \in FIRST(\alpha) \Rightarrow a \in FOLLOW(A) \Rightarrow a \in SELECT(A \rightarrow \beta)$$

$$A \Rightarrow A\alpha \Rightarrow \beta\alpha \Rightarrow \alpha \Rightarrow a \in SELECT(A \rightarrow A\alpha)$$

$$2. FIRST(\alpha) = \{\lambda\}$$

$$SELECT(A \rightarrow A\alpha)$$

$$\lambda \in FIRST(A\alpha) \Rightarrow FOLLOW(A) \subseteq SELECT(A \rightarrow A\alpha)$$

$$A \Rightarrow A\alpha \Rightarrow \beta\alpha \Rightarrow \alpha \Rightarrow \lambda$$

$$\emptyset \neq FOLLOW(A) \subseteq SELECT(A \rightarrow A\alpha) \cap SELECT(A \rightarrow \beta)$$

Всё умеем. Давайте построим анализатор. Для селекта нужен фёрст

Алгоритм построения множества FIRST для символьной строки

$$G = \langle \Sigma, \Gamma, P, S \rangle$$

$$\forall a \in \Sigma : FIRST(a) = \{a\}$$

$$\forall A : (A \rightarrow \lambda) \in P : FIRST(A) = \{\lambda\}$$

Пока множество FIRST не стабилизируется, повторяем:

$$\forall (A \rightarrow X_1 \dots X_n) \in P, n > 0$$

$$i = 1;$$

$$(*) FIRST(A) = FIRST(A) \cup (FIRST(X_i) \cap \Sigma)$$

$$\text{если } \lambda \in FIRST(X_i)$$

$$\text{если } i < n$$

$$i + +; \text{ перейти к } (*)$$

$$\text{иначе } FIRST(A) = FIRST(A) \cup \{\lambda\}$$

Простыми словами:

Для всех терминалов положим в FIRST сам этот терминал

Для всех аннулирующих нетерминалов положим в FIRST лямбду

Пока множество FIRST не стабилизируется:

- для каждого нетерминала рассмотрим его правила
- положим в FIRST нетерминала FIRST первого символа правой части
- если этот символ может аннулироваться ($\lambda \in FIRST(X_1)$), то нужно двигаться дальше по правой части, пока не дойдём до конца или пока не встретим неаннулирующийся символ

Алгоритм построения множества FOLLOW для символьной строки

Находим правые части, куда входит данный нетерминал: $B \rightarrow \alpha A \beta$. Сначала надо посмотреть на $FIRST(\beta) \setminus \{\lambda\} \subseteq FOLLOW(A)$. Если β аннулируется ($\{\lambda\} \in FIRST(\beta)$):

$$S \Rightarrow^* \gamma_1 B \gamma_2 \Rightarrow \gamma_1 \alpha A \beta \gamma_2 \Rightarrow \gamma_1 \alpha A \gamma_2 - FIRST(\gamma_2) \subseteq FOLLOW(B)$$

$$FOLLOW(S) = \{\vdash\}$$

Пока множество FOLLOW не стабилизируется, повторяем:

$$\forall (A \rightarrow X_1 \dots X_n) \in P, n > 0$$

если ($X_n \in \Gamma$)

$$FOLLOW(X_n) = FOLLOW(X_n) \cup FOLLOW(A)$$

$$i = n - 1;$$

ann = true; *флаг, что хвост аннулируемый*

(*) если $i > 0$ и $X_i \in \Gamma$ и X_i — терминал

$$FOLLOW(X_i) = FOLLOW(X_{i+1}) \cup (FIRST(X_{i+1} \dots X_n) \cap \Sigma)$$

пересечение нужно, чтобы в FOLLOW не попала λ

если $\lambda \notin FIRST(X_{i+1})$

ann = false

если ann == true и $X_i \in \Gamma$

$$FOLLOW(X_i) = FOLLOW(X_i) \cup FOLLOW(A)$$

$i - -$; перейти к (*)

Простыми словами:

1. $FOLLOW(S) = \{\vdash\}$

2. Для правил вида $A \rightarrow \alpha B \beta$: $FOLLOW(B) = FIRST(\beta) \setminus \{\lambda\}$

Сразу за B может следовать всё, что выводится из β первым символом

3. Для правила вида $A \rightarrow \alpha B \beta$, где $\lambda \in FIRST(\beta)$ и $A \rightarrow \alpha B$:

$$FOLLOW(B) = FOLLOW(A)$$

Если B — крайний правый символ, то сразу за ним может следовать то же, что и за A

Пример

$$E \rightarrow TE' \quad (1)$$

$$E' \rightarrow +TE' \mid \lambda \quad (2) \quad (3)$$

$$T \rightarrow FT' \quad (4)$$

$$T' \rightarrow *FT' \mid \lambda \quad (5) \quad (6)$$

$$F \rightarrow (E) \mid x \quad (7) \quad (8)$$

	FIRST	FOLLOW
E	(, ×	¬,)
E'	λ, +	¬,)
T	(, ×	¬, +,)
T'	λ, *	¬, +,)
F	(, ×	¬, +, *,)

$$SELECT(A \rightarrow \alpha) = FIRST(\alpha) \setminus \{\lambda\}$$

$$\text{Если } \lambda \in FIRST(\alpha) \text{ то } SELECT(A \rightarrow \alpha) \cup = FOLLOW(A)$$

Нетерминал	Правило	SELECT
E	(1)	(, x
E'	(2)	+
E'	(3)	¬,)
T	(4)	(, x
T'	(5)	*
T'	(6)	+, ¬,)
F	(7)	(
F	(8)	x

Множества SELECT для каждого из нетерминалов не пересекаются, значит, это LL(1)-грамматика.

Построение нисходящего анализатора по LL(1)-грамматике

$$G = (\Sigma, \Gamma, P, S) \text{ — LL(1)}$$

$M = (\bar{\Sigma}, \bar{\Gamma}, \delta, s)$ — ДМПА, распознающий $L(G)$

$$\bar{\Sigma} = \Sigma \cup \{\neg\}$$

$$\bar{\Gamma} = \Gamma \cup \Sigma \cup \nabla$$

δ :

1. $(\nabla, \neg) \rightarrow \checkmark$
2. $\forall a \in \Sigma : (a, a) \rightarrow (\lambda, \rightarrow)$
3. $\forall A \in \Gamma, \forall (A \rightarrow \beta) \in P : \forall a \in SELECT(A \rightarrow \beta) : (A, a) \rightarrow (\beta, \neg)$

Простыми словами:

1. Заканчиваем работу успешно, если стек пуст, а входная строка дочитана до конца
2. Читаем символ входной строки только тогда, когда, когда на вершине стека — этот же символ
3. Для каждого символа входной строки будем класть в стек правую часть того правила, в SELECT которого входит этот символ

Обработка синтаксических ошибок

	x	$+$	$*$	$($	$)$	\neg
E - новое подвыражение	TE'	2	2	TE'	4	2
E' - ожидание оператора и слагаемого	1	$+TE'$		1	λ	λ
T - начало операнда	FT'	2	2	FT'	4?	2
T' - ожидание оператора и множителя	1	λ	$*FT'$	1	λ	λ
F - операнд, из него выводится x или подвыражение	x	2	2	(E)	4	2?
$)$	3	3	3	3	λ, \rightarrow	3
∇					4	\checkmark

Команды обработки терминалов в стеке не указаны в таблице, но подразумеваются. Не пишем, чтобы не раздувать таблицу. Закрывающая скобка — только для того, чтобы указать ошибки

Обрабатывать ошибки — это не только сообщать о них, но и продолжать работу.

Ошибка — попадание в пустую клетку управляющей таблицы.

Как бороться?

- режим паники — пропускать нехорошие входные символы;

ждать, пока не увидим терминал из множества FIRST или FOLLOW для нетерминала из стека. В первом случае мы не снимаем A со стека и делаем переход по нему. Во втором случае нужно снять A со стека, так как блок закончился.

- снимать элементы из стека

Арифметические ошибки

1. Пропущен оператор — добавить $+$;
2. Пропущен операнд — добавить x ;
3. Незакрытая левая скобка — закрыть;
4. Преждевременная правая скобка — удалить.

1. $)(x + x)(*$

Пропустили первую скобку

$E\triangledown$

$TE'\triangledown$

$FT'E'\triangledown$

$E)TE'\triangledown E)$ — генерация подвыражения

...

$TE'\triangledown$, видим левую скобку, значит, нужно добавить плюс. Добавляем, видим плюс, сокращаем его.

$E'\triangledown$

$TE'\triangledown$ снова видим скобку

$FT'E'\triangledown$

$E)T'E'\triangledown$ видим умножение и пропущенный операнд. Вставим его

...

2. $(x + x))(*x$

$E\triangledown$

$TE'\triangledown$

$FT'E'\triangledown$

$T'E'\triangledown$ — где то тут смотрим на лишнюю скобку, но пока не можем понять, что это ошибка

$E'\triangledown$

\triangledown тут мы ошибку поймаем, но дальше продолжить не сможем, т.к. стек пуст

LL(k)-грамматики

FIRST

Опр. $FIRST_k(\alpha) \subseteq \Sigma \cup \{\lambda\}$:

$w \in FIRST_k(\alpha) \iff \alpha \Rightarrow^* v$, где:

1. $|v| < k, w = v$ — могут быть цепочки **меньшей** чем k длины!
2. $|w| \geq k, |w| = k, w$ - префикс v

Опр. G — $LL(k)$ -грамматика, если из существования двух выводов:

$S \Rightarrow^* wA\alpha \Rightarrow w\beta\alpha \Rightarrow^* wv$

$S \Rightarrow^* wA\alpha \Rightarrow w\gamma\alpha \Rightarrow^* wu$

где $FIRST_k(u) = FIRST_k(v)$, следует, что $\beta = \gamma$.

Отсюда можно сделать ложный вывод, что если мы обобщим **FIRST** и **FOLLOW**, и попытаемся перенести определение из $LL(1)$, то мы получим то же самое. Однако, это необязательно так.

Классы грамматик

$LL(1) \subset LL(2) \subset \dots \subset LL(k) \subset LL(k+1)$

$S \rightarrow a^k b | a^k c$ — $LL(k+1)$, но не $LL(k)$

Пример

$S \rightarrow abA | \lambda$

$A \rightarrow Saa | b$

Не $LL(1)$, для первых двух правил пересекаются селекты:

$FOLLOW(S) = \{a, \vdash\}$

$SELECT(S \rightarrow abA) \cap SELECT(S \rightarrow \lambda) = \{a\}$

Надо уметь раскрывать нетерминал, видя два символа. Проблема, когда на вершине стека аксиома. Такое бывает только два раза: в самом начале, когда мы знаем, что нам применять, либо после применения третьего правила.

$S \Rightarrow^* wA\alpha \Rightarrow wSaa\alpha$ — S либо аннулируется, либо развернётся:

$\Rightarrow waa\alpha$ — если видимо это, то применяем второе правило;

$\Rightarrow wabAaaa$ — если видим это, то применяем первое.

Определить это можем по двум символам, значит, это $LL(2)$.

■

Определить, является ли грамматика $LL(k)$ грамматикой для **какого-нибудь** k — алгоритмически неразрешима. Но определить это для **конкретного** k можно.

КС языки распознаются НМПА. А LL — детерминированными. Наверное, есть и не LL язык. И это так!

$\{a^k 0 b^k\} \cup \{a^k 1 b^{2k}\}$

Метод рекурсивного спуска

Если в грамматике существует число, ограничивающее длину вывода, то можно эмулировать вывод, пускай даже рекурсивный. Надо перебрать те правила, которые есть в SELECT для терминала.

- есть возможность отката налево (в Шуре этого нет!) и вверх и обстригания дерева.

```
1 function A() {
2     const rules = [все правила вида  $A \rightarrow X_1X_2 \dots X_k$ ];
3     let prevPtr = PTR; // указатель на текущий символ входной строки
4     for (let rule in rules) {
5         const x = rule.rightPart;
6         const k = rule.rightPart.size;
7         let hasErrors = false;
8         for (let i = 0; i < k && !hasErrors; i++) {
9             if (x[i].IsNonterminal) {
10                 hasErrors = x_i(); // вызов процедуры  $X_i$ 
11             } else if (x[i] === input[prevPtr]) {
12                 prevPtr++; // переходим к следующему символу
13             } else {
14                 hasErrors = true;
15             }
16         }
17
18         if (hasErrors) {
19             prevPtr = PTR; // откатываемся обратно
20             continue; // пробуем другие правила
21         }
22
23         PTR = prevPtr; // смогли раскрыть правило без ошибок, двигаемся
24         дальше
25         return false;
26     }
27
28     // перебрали все правила, ни одно не подошло, ошибка
29     return true;
30 }
```

Код выше написан из головы и из Dragon book. Нужны чьи-нибудь конспекты и внимательный взгляд