Нисходящий анализ

Левая рекурсия

Грамматика называется **непосредственно леворекурсивной**, если $\exists (A \to A\alpha) \in P$. 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.

$A \to A\alpha_1 | \dots | A\alpha_n | \beta_1 | \dots | \beta_m$ $\forall i : \beta_i$ не начинается с A

A' — новый нетерминал

Заменим все рекурсивные правила на такую группу

$A \to \beta_1 A' | \dots | b_m A'$

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

> Левая рекурсия - это накопление альф, 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 $A_i \to A_j\alpha$ заменить на $A_i \to \beta\alpha$, где $(A_j \to \beta) \in P$
  Устранить непосредственную рекурсию для A_i

Доказательство корректности — индукция по i

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

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

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

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

ШИ $A_i \to A_m\alpha$, где $m < i$

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

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

$S \to Aa|AB|B$

$A \to SB|ac$

$B \to Ac|b$

Надо перенумеровать:

$S_1 \to A_2a|A_2B_3|B_3$

$A_2 \to S_1B_3|ac$ [зачёркнуто после 2 итерации]

$B_3 \to A_2c|b$

После внутреннего цикла:

$A_2 \to A_2aB_3|A_2B_3B_3|B_3B_3|ac$ [зачёркнуто после 2 итерации]

Добавим A':

$A_2 \to B_3B_3A'|acA'$

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

Третья итерация:

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

Устраним непосредственную рекурсию:

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

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

Готово.

Левая факторизация

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

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 пройтись на неопределённую глубину, чтобы понять, какое правило было применено.

> $S \to if(B)S|if(B)S\ elseS$ — 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 $A \to \beta\alpha_1|\beta\alpha_2$ на:

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

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

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

$A \to Bc|b$

$B \to aa$

$C \to aA$

Самый длинный общий префикс — Ab

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

$D \to c|B$

Самый длинный общий префикс - A

$S \to AE$

$E \to bD|C|BB$

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.

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

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

$u|v$

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 выводится 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

$S \to AC$

$A \to abC|bB$

$B \to b$

$C \to c|\lambda$

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

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

FOLLOW

Опр. $FOLLOW(A) \subseteq \Sigma \cup \{\dashv\}$:
$a \in FOLLOW(A) \iff S \Rightarrow^* \alpha A a \beta$
$\dashv \in FOLLOW(A) \iff S \Rightarrow^* \alpha A$

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

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

SELECT

Опр. $SELECT(A \to \alpha)$:

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

> Множество выбора правил.

> $SELECT(A \to \alpha) = FIRST(\alpha FOLLOW(A))$

$LL(1)$-грамматика

Опр. $LL(1)$-грамматика:
$\forall A \in \Gamma : \forall (A \to \beta), (A \to \alpha) \in P$:
$SELECT(A \to \alpha) \cap SELECT(A \to \beta) = \varnothing$

Множество $SELECT(A \to \alpha)$ хранит в себе множество символов, увидя который, нужно применить правило $A \to \alpha$. 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

$E \to E + T|T$

$T \to T * F|F$

$F \to (E)|x$

Устраним рекурсию

$E \to TE'$

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

$T \to FT'$

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

$F \to (E)|x$

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

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

> *23.04.2019*

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

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

$(A \to \beta) \in P$, где $\beta[1] \neq A$

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

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

   ! $FIRST(A) = \bigcup\limits_{(A \to \beta) \in P} FIRST(\beta)$
2. $FIRST(\beta) = \{\lambda\}$ — из β выводится только λ
   1. $a \in FIRST(\alpha) \Rightarrow a \in FOLLOW(A) \Rightarrow a \in SELECT(A \to \beta)$

      $A \Rightarrow A\alpha \Rightarrow \beta\alpha \Rightarrow \alpha \Rightarrow a \in SELECT(A \to A\alpha)$
   2. $FIRST(\alpha) = \{\lambda\}$

      $SELECT(A \to A\alpha)$

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

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

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

Всё умеем. Давайте построим анализатор. Для селекта нужен фёрст

Алгоритм построения множества FIRST для символьной строки

$G = <\Sigma, \Gamma, P, S>$

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

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

Пока множество FIRST не стабилизируется, повторяем:

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

$i = 1;$

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

если $\lambda \in FIRST(X_i)$

  если $i < n$

    $i++$; перейти к (*)

  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: $B \to \alpha A\beta$. Сначала надо посмотреть на $FIRST(\beta) \setminus \{\lambda\} \subseteq FOLLOW(A)$. Если β аннулируется ($\{\lambda\} \in FIRST(\beta)$):
>
> $S \Rightarrow^* \gamma_1 B\_\gamma_{2\_} \Rightarrow \gamma_1\alpha A\beta\_\gamma_{2\_} \Rightarrow \gamma_1\alpha A\_\gamma_{2\_} - FIRST(\gamma_2) \subseteq FOLLOW(B)$

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

Пока множество FOLLOW не стабилизируется, повторяем:

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

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. $FOLLOW(S) = \{\dashv\}$
> 2. Для правил вида $A \to \alpha B\beta$: $FOLLOW(B) = FIRST(\beta) \setminus \{\lambda\}$
>
>   *Сразу за B может следовать всё, что выводится из β первым символом*
> 3. Для правила вида $A \to \alpha B\beta$, где $\lambda \in FIRST(\beta)$ и $A \to \alpha B$: $FOLLOW(B) = FOLLOW(A)$

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

Пример

$E \overset{(1)}{\to} TE'$

$E' \to \overset{(2)}{+TE'} | \overset{(3)}{\lambda}$

$T \overset{(4)}{\to} FT'$

$T' \to \overset{(5)}{*FT'} | \overset{(6)}{\lambda}$

$F \to \overset{(7)}{(E)} | \overset{(8)}{x}$

	FIRST	FOLLOW
E	$(, \times$	$\dashv,)$
E'	$\lambda, +$	$\dashv,)$
T	$(, \times$	$\dashv, +,)$
T'	$\lambda, *$	$\dashv, +,)$
F	$(, \times$	$\dashv, +, *,)$

> $SELECT(A \to \alpha) = FIRST(\alpha) \setminus \{\lambda\}$
>
> Если $\lambda \in FIRST(\alpha)$ то $SELECT(A \to \alpha) \cup = FOLLOW(A)$

Нетерминал	Правило	SELECT
E	(1)	$(, x$
E'	(2)	$+$
E'	(3)	$\dashv,)$
T	(4)	$(, x$
T'	(5)	$*$
T'	(6)	$+, \dashv,)$
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

$G = (\Sigma, \Gamma, P, S)$ — LL(1)

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

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

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

δ:

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

> **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	$+$	*	(	)	$\dashv$
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	$\lambda, \to$	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

$E\nabla$
$TE'\nabla$
$FT'E'\nabla$
$E)TE'\nabla$ E) — генерация подвыражения
...
$TE'\nabla$, видим левую скобку, значит, нужно добавить плюс. Добавляем, видим плюс, сокращаем его.
$E'\nabla$
$TE'\nabla$ снова видим скобку
$FT'E'\nabla$
$E)T'E'\nabla$ видим умножение и пропущенный операнд. Вставим его
...

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

$E\nabla$
$TE'\nabla$
$FT'E'\nabla$
$T'E'\nabla$ — где то тут смотрим на лишнюю скобку, но пока не можем понять, что это ошибка
$E'\nabla$
∇ тут мы ошибку поймаем, но дальше продолжить не сможем, т.к. стек пуст

LL(k)-грамматики

FIRST

<u>Опр</u>. $FIRST_k(\alpha) \subseteq \Sigma^*$:
$w \in FIRST_k(\alpha) \iff \alpha \Rightarrow^* v$, где:

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

<u>Опр</u>. 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

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

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

Пример

$S \to abA|\lambda$

$A \to Saa|b$

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

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

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

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;
$\Rightarrow wabAaa\alpha$ — 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.

```
function A() {
    const rules = [все правила вида A → X₁X₂...Xₖ];
    let prevPtr = PTR; // указатель на текущий символ входной строки
    for (let rule in rules) {
        const X = rule.rightPart;
        const k = rule.rightPart.size;
        let hasErrors = false;
        for (let i = 0; i < k && !hasErrors; i++) {
            if (X[i].IsNonterminal) {
                hasErrors = X_i(); // вызов процедуры Xᵢ
            } else if (X[i] === input[prevPtr]) {
                prevPtr++;  // переходим к следующему символу
            } else {
                hasErrors = true;
            }
        }

        if (hasErrors) {
            prevPtr = PTR; // откатываемся обратно
            continue;      // пробуем другие правила
        }

        PTR = prevPtr; // смогли раскрыть правило без ошибок, двигаемся дальше
        return false;
    }

    // перебрали все правила, ни одно не подошло, ошибка
    return true;
}
```

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