Контекстно-свободные грамматики. Нормальная форма Хомского. Общие методы разбора.

Контекстно-свободные грамматики

Для чего они нужны? Пример – язык палиндромов $L_{pal} = \{w : w = w^R\}$. Можно показать, что он не является регулярным, рассмотрев палиндром $L_{pal} = \{0^n 1 0^n\}$ и применив лемму о разрастании. Не все полезные языки – регулярные. Нужен более широкий класс.

Запишем правила определения палиндромов в виде КС-грамматики:

1. $P \to \varepsilon$
2. $P \to 0$
3. $P \to 1$
4. $P \to 0P0$
5. $P \to 1P1$

Формально: КС-грамматика G – это четверка $G = (V, T, P, S)$, где V – множество переменных, T – терминалов, P – продукций, S – стартовый символ. Для языка палиндромов: $G_{pal} = \{\{P\}, \{0,1\}, A, P\}$, где A – множество продукций (1) – (5). Другой пример:

$E \to I$,
$E \to E + E$, $E \to E * E$, $E \to (E)$,
$I \to a$, $I \to b$, $I \to Ia$, $I \to Ib$, $I \to I0$, $I \to I1$.

Два способа убедиться, что цепочка принадлежит КС-грамматике: «от тела к голове» (рекурсивный вывод) – объединяем цепочки принадлежащие грамматике по правилам, и «от головы к телу» (порождение) – разворачиваем стартовый символ. Первый способ требует введения символа отношения $\Rightarrow$. Если $\alpha A \beta$ – цепочка из терминалов и переменных, где A – переменная, $A \to \gamma$ – продукция из G, тогда мы говорим, что $\alpha A \beta \underset{G}{\Rightarrow} \alpha \gamma \beta$. Пример порождения: $E \Rightarrow E * E \Rightarrow I * E \Rightarrow a * E \Rightarrow a * (E) \Rightarrow a * (E + E)$ $\Rightarrow a * (I + E) \Rightarrow a * (a + E) \Rightarrow a * (a + I) \Rightarrow a * (a + I0) \Rightarrow a * (a + I00) \Rightarrow a * (a + b00)$.

Рекурсивный вывод осуществляется в обратном порядке. Символ $\overset{*}{\Rightarrow}$ используется для сокращения порождения: $E \overset{*}{\Rightarrow} a * (a + b00)$.

Левые и правые порождения. Для ограничения числа выборов в процессе порождения потребуем, чтобы на каждом шаге заменялась самая левая (правая) переменная одним из тел продукции. Для указания такого левого (правого) порождения используются символы: $\underset{lm}{\Rightarrow}$ и $\underset{lm}{\overset{*}{\Rightarrow}}$ ($\underset{rm}{\Rightarrow}$ и $\underset{rm}{\overset{*}{\Rightarrow}}$).

Язык, задаваемый грамматикой. Если $G = (V, T, P, S)$, то язык, задаваемый грамматикой G, обозначается $L(G)$ и представляет собой множество терминальных цепочек, порождаемых из стартового символа: $L(G) = \{w \in T^* \mid S \underset{G}{\overset{*}{\Rightarrow}} w\}$. Такой язык называется контекстно-свободным.

Деревья разбора. *Деревья разбора* для G – это деревья со следующими свойствами: 1) каждый внутренний узел отмечен переменной из V; 2) каждый лист отмечен либо переменной, либо терминалом, либо ε (в этом случае он единственный сын своего родителя); 3) если внутренний узел отмечен A, и его сыновья отмечены слева направо $X_1, X_2, \ldots, X_k$, то $A \to X_1 X_2 \ldots X_k$ является продукцией в P.

Теорема. При любой грамматике $G = (V, T, P, S)$ следующие утверждения равносильны: 1) процедура рекурсивного вывода определяет, что цепочка w принадлежит языку переменной A; 2) $A \underset{lm}{\overset{*}{\Rightarrow}} w$; 3) $A \underset{rm}{\overset{*}{\Rightarrow}} w$; 4) $A \overset{*}{\Rightarrow} w$.

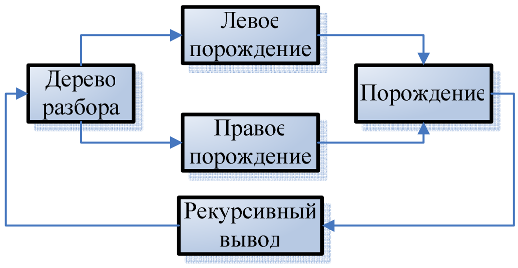

Рис. 1. Идея доказательства

Нормальная форма Хомского

Зачем нужна нормальная форма? Например, для эффективного разбора – проверки, что некоторое слово принадлежит КС-языку. Оказывается каждый КС-язык (без ε) порождается грамматикой, не имеющей *бесполезных символов*, все продукции которой имеют одну из двух форм: $A \to BC$ или $A \to a$, где A, B, C – переменные, a – терминал. Такая форма грамматик называется **нормальной формой Хомского (НФХ)**. Для преобразования в неё надо:

1. Удалить *ε-продукции*, т.е. продукции вида $A \to \varepsilon$ для некоторой переменной A.
2. Удалить *цепные продукции*, т.е. продукции вида $A \to B$ с переменными A, B.
3. Удалить *бесполезные символы*, т.е. недостижимые из стартового, либо не участвующие в порождении цепочек из терминалов.
4. Сделать так, чтобы все тела продукций длины 2 и более состояли только из переменных. Для этого достаточно для каждого терминала a, встречающегося в такой продукции создать новую переменную A, заменить переменную в исходной продукции и добавить новую продукцию $A \to a$.
5. Разбить тела длины 3 и более на группу продукций, тело каждой из которых состоит из двух переменных. Для этого достаточно ввести новые переменные и сделать замены.

При этом язык, задаваемый грамматикой, не изменяется. Порядок первых трех действий имеет значение, т.к. (2) и (3) не приводят к появлению *ε-продукций*, а (3) не приводит к появлению цепных продукции. Все удаления выполняются индуктивно. Доказательства корректности (все желаемое удалили, ничего лишнего не удалили) производятся просто по индукции, поэтому будет приведен только один пример такого доказательства.

Удаление ε-продукций. Сначала обнаружим ε-порождающие переменные, т.е $A \stackrel{*}{\Rightarrow} \varepsilon$. Если $A \to \varepsilon$, то A очевидно ε-порождающая переменная. Если в G есть продукция $B \to C_1C_2\ldots C_k$, где каждая C_i ε-порождающая, то B также ε-порождающая. Приведем **доказательство** того, что в любой грамматике ε-порождающими являются только продукции найденные описанным алгоритмом. Любая переменная, найденная алгоритмом, ε-порождающая по построению. Покажем, что любая ε-порождающая продукция A грамматики G будет найдена алгоритмом. Применим индукцию по длине кратчайшего (по числу шагов) порождения $A \stackrel{*}{\Rightarrow} \varepsilon$. Первый шаг $A \Rightarrow C_1C_2\cdots C_k$, где каждый C_i порождает ε за число шагов меньшее n. Значит, по индуктивному предположению все C_i будут найдены алгоритмом, а следовательно будет найдена и переменная A.

Определив все ε-порождающие переменные, выполним следующее. Если $A_1, A_2,.., A_k \stackrel{*}{\Rightarrow} \varepsilon$ и грамматика содержит продукцию вида $B \to CA_1DA_2...XA_k$, то можно заменить ее на 2^k продукций вида: $B \to CA_1DA_2...XA_k$, $B \to CD...XA_k$, $B \to CA_1D...XA_k$, $B \to CA_1DA_2...XA_k$, $B \to CD...X$, в зависимости от того, будет ли каждая из A_i выводить ε. После всех таких замен удалим все *ε-продукции*.

Удаление цепных продукций. Определим все пары (A, B), для которых $A \overset{*}{\Rightarrow} B$ лишь с использованием цепных продукций. (A, A) – цепная пара. Если (A, B) – цепная и $B \to C$, то (A, C) – цепная. После определения всех таких пар, заменим все $A \Rightarrow B_1 \Rightarrow ... \Rightarrow B_n \Rightarrow \alpha$ с нецепной продукцией $B_n \Rightarrow \alpha$, на продукцию $A \to \alpha$. После этого удалим все цепные продукции.

Удаление бесполезных символов. Найдем непорождающие символы. Для этого по индукции пометим все порождающие. Удалим все непорождающие и все продукции, в которых они участвуют. Далее найдем все неждостижимые символы (индуктивно пометив достижимые) и аналогично поступим с ними.

Оценка времени работы. Пусть грамматика содержит n продукций. Выясним, за какое время грамматика приводится к НФХ. Удаление цепных продукций: $O(n^2)$, удаление бесполезных символов: $O(n)$, время (4) и (5) действия: $O(n)$. Если перед удалением **ε-продукций** выполнить действие (5), то экспоненциальное время удаления таких продукций превратится (за счет короткой длины продукций, равной двум) в линейное. Итого, время преобразования – $\boldsymbol{O(n^2)}$.

Общие методы разбора

Дано слово w, длины n. Разбор – проверка принадлежности слова КС-языку. Приведем грамматику к нормальной форме Хомского. Далее, простой способ разбора – перебрать все двоичные деревья разбора для слова длины n. Недостаток – экспоненциальное время по n, потому что таких деревьев много.

Существует более эффективный *алгоритм Кока-Янгера-Хасами*. Этот алгоритм работает за $O(n^3)$. Идея – динамическое программирование. Обозначим за f_{ij} множество переменных A, для которых $A \overset{*}{\Rightarrow} a_i a_{i+1} \cdots a_j$. Тогда ответ на задачу сводится к проверке принадлежности S множеству f_{1n}, т.е. можно ли из стартового символа вывести всё слово.

Инициализация. Пусть $f_{ij} = \varnothing$. Для каждого $i = 1..n$ и продукции вида $A \to a_i$ добавим продукцию A во множество f_{ii}.

Вычисление. Будем проводить динамику по длине подпоследовательности: $j - i$. Допустим, надо найти множество переменных, порождающих слово $a_i a_{i+1} \cdots a_j$. При этом нам известны множества переменных порождающих любой префикс и суффикс этого слова (не совпадающий с ним самим). Разобьем слово некоторым способом на префикс $a_i \cdots a_k$ и суффикс $a_{k+1} \cdots a_j$. Тогда, если f_{ik} содержит некоторую переменную B, а f_{k+1j} содержит некоторую переменную C, и грамматика G содержит продукцию $A \to BC$, то множество f_{ij} будет содержать переменную A. Проведя разбиение строки на префикс и суффикс всеми возможными способами, перебрав все пары переменных (B, C) из множеств соответствующих префиксу и суффиксу, и добавив во множество f_{ij} все такие A, что $A \to BC$, мы полностью вычислим f_{ij}. Формула: $f_{ij} = \bigcup_{k=i}^{j-1} \{A \mid A \to BC, B \in f_{ik}, C \in f_{k+1j}\}$.

Анализ времени работы. Вычисление f_{ij} производится для всех пар $i, j = 1..n$. И того $O(n^2)$. Вычисляя некоторое f_{ij}, мы перебираем все разбиения на суффиксы и префиксы, всего их $O(n)$. Для каждого разбиения перебираются все пары переменных (B, C) из соответствующих множеств и проверяется, принадлежит ли продукция $A \to BC$ грамматике для всех символов A. И того, $O(k^3)$, где k – число переменных в грамматике. Но поскольку k от длины слова n не зависит, то $O(k^3) = O(1)$.

В результате приходим к $\boldsymbol{O(n^3)}$**.**