3.8. Контекстно-свободные грамматики. Эффективные методы разбора: LL и LR грамматики.

Определение

КС-грамматика G – это четверка: $G = \{V, T, P, S\}$

V – множество переменных (нетерминалов),

T – множество токенов (терминалов),

P – множество продукций вида $A \to \alpha_1\alpha_2\ldots\alpha_k$, где A – нетерминал, каждый из α_i – терминал либо нетерминал. Правая часть продукции может быть пустой (ε).

S – стартовый символ.

Пример 1: язык палиндромов $G_{pal} = \{\{P\}, \{0,1\}, A, P\}$, где A – множество продукций:

$P \to \varepsilon$
$P \to \mathbf{0}$
$P \to \mathbf{1}$
$P \to \mathbf{0}P\mathbf{0}$
$P \to \mathbf{1}P\mathbf{1}$

Пример 2:

$E \to I$,
$E \to E + E$, $E \to E * E$, $E \to (E)$,
$I \to \mathbf{a}$, $I \to \mathbf{b}$, $I \to I\mathbf{a}$, $I \to I\mathbf{b}$, $I \to I\mathbf{0}$, $I \to I\mathbf{1}$.

КС-язык – язык, порождаемый КС-грамматикой.
NB любой регулярный язык (т.е. задаваемый регулярным выражением) является КС.

Общие методы разбора:

1. Приведение к нормальной форме Хомского.
2. Алгоритм Кока-Янгера-Хасами.

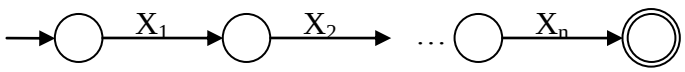
(см 3.7)

Нисходящий анализ

Рекурсивный нисходящий анализ

Для каждого нетерминала строится своя диаграмма переходов, по сути конечный автомат, следующим образом:

1. Изначально есть стартовое и конечное состояние
2. Для каждой продукции $A \to X_1X_2\ldots X_n$ добавляются переходы:



Разбор:

Изначально алгоритм находится в стартовом состоянии диаграммы стартового нетерминала.

Пусть текущее состояние s.

- Если есть переход из s по терминалу a, и очередной входной символ – a, то производится этот переход, и алгоритм переходит к следующему входному символу.
- Если есть переход из s по нетерминалу A, то входной символ не пропускается, а рекурсивно вызывается алгоритм на диаграмме состояний A.
- Если есть ε-переход из s, то очередной символ не меняется, и производится переход.

3.8

Как дошли до конечного состояния – возвращаемся из рекурсии. Если дошли до конечного состояния исходной диаграммы переходов (на верхнем уровне рекурсии), то разобрали все слово.

Проблема: не всегда однозначно понятно, какой делать переход. Приходится проверять разные варианты и откатываться в случае неудачи. Если нам повезло, и в наших диаграммах переходов все однозначно, то получили *предикативный* анализатор (т.е. такой, который сразу по очередному символу знает, какие продукции применять).

Нерекурсивный предикативный анализ

Использует таблицу разбора – функцию M : V x T → P ∪ {**error**}. Т. е. по каждой паре {нетерминал, терминал} таблица хранит то, какую продукцию применить, либо специальный маркер ошибки.

Алгоритм:

Изначально в стек кладется $S (S – стартовый символ, на верхушке, $ – специальный символ конца строки, на дне). Далее смотрим текущий символ на вершине стека X и очередной символ входной строки a.

1. Если X = a = $ – конец.
2. Если X = a != $ – снять со стека X и перейти к следующему символу входного потока.
3. Если X – терминал, но X != a, то ошибка.
4. Если X – нетерминал, то смотрим M[X,a]. Если там **error**, то строка не принадлежит языку, ошибка. Иначе там продукция вида $X \to Y_1 Y_2 \dots Y_k$. Тогда снимаем со стека X и кладем туда $Y_k \dots Y_2\ Y_1$ (Y_1 теперь на вершине).

Как построить таблицу:

Сначала вычислим множества FIRST(X) для всех символов и FOLLOW(X) для всех нетерминалов.

FIRST(α) – множество всех терминалов, с которых могут начинаться строки, выводимые из α.

FOLLOW(X) – множество всех терминалов, которые могут располагаться непосредственно справа от X в любой сентенциальной форме (то есть в промежуточной строчке вывода)

1. Для каждой продукции A→α делаем шаги 2 и 3
2. Для каждого терминала a из FIRST(α) добавляем в ячейку M[A,a] правило A→α.
3. Если в FIRST(α) есть ε, то для каждого терминала b из FOLLOW(A) добавим в ячейку M[A,b] правило A→α. Если при этом FOLLOW(A) есть $, то добавим в ячейку M[A,$] правило A→α.
4. Во все остальные ячейки запишем **error**.

LL(1)-грамматика – такая грамматика, для которой вышеприведенный алгоритм построит однозначную таблицу. Т.е. для нее нисходящий разбор будет работать.

LL(1)-грамматика всегда однозначна и леворекурсивна (но не наоборот), поэтому чтобы попытаться из грамматики сделать LL(1) нужно устранить левую рекурсию и провести левую факторизацию (не факт, что это поможет, но на практике помогает).

Первое L значит, что парсим слева направо, второе L – что получаем левое порождение, 1 означает, что достаточно одного символа, чтобы понять, какую продукцию применять.

3.8

Восходящий анализ

ПС-анализ

ПС – «перенос/свертка» (shift/reduce)

Общая идея:
Читаем символы и кладем их в стек (перенос). Как только находим что-то похожее на правую часть продукции на вершине стека – заменяем это на соответствующую левую часть (свертка). В конце концов должны получить стартовый символ.

Проблемы: в ходе решения бывает непонятно по состоянию стека и входному символу что делать:

1. Возможно сделать перенос очередного символа или свертку. (конфликт *перенос/свертка*)
2. Возможно сделать свертку по различным продукциям (конфликт *свертка/свертка*)

Для разных грамматик есть разные модификации этого метода.

Синтаксический анализ приоритета операторов
Операторная грамматика – такая грамматика, у всех продукций которой правые части не ε и в них нет двух нетерминалов подряд.

LR-анализ
Имеем:

- Входной поток
- Стек
- Таблица синтаксического анализа

Стек всегда имеет вид $s_0X_1s_1\ldots X_ms_m$, где X – символы грамматики, s – *состояния*. Состояние и очередной входной символ являются индексом таблицы синтаксического анализа.

Таблица состоит из двух частей: функции действий (action) и функции перехода (goto).

Алгоритм: на каждом шаге берем текущее состояние на вершине стека s_m, текуший входной символ a_i. Смотрим в таблицу action[s_m ,a_i]. Там может быть 4 варианта:

1. «перенос s» Выполняется перенос: в стек добавляется a_i и s, переходим к следующему символу
2. «свертка A→β» Выполняется свертка по продукции A→β. последние 2r ячеек удаляются из стека, туда добавляется A и s=goto[s_{m-r},A], где r – длина β.
3. «допуск» Анализ завершен
4. «ошибка»

Как построить таблицу?
Есть разные методы:

1. SLR-метод (Simple LR)
2. Канонический LR метод
3. LALR-метод (lookahead LR)

Здесь так же как в LL анализе используются множества FIRST и FOLLOW (а так же много разных других вещей, см. в драконе).

3.8

Канонический метод – самый общий и сильный, но в нем получаются самые большие, громоздкие таблицы.

LR(1)-грамматика (или просто LR-грамматика) – грамматика, для которой можно построить таблицу синтаксического анализа каноническим LR-методом.

L значит, что парсим слева направо, R – что получаем правое левое порождение, 1 означает, что достаточно одного символа, чтобы понять, какую продукцию применять.

Аналогичны понятия SLR- и LALR-грамматики. Все они являются частным случаем LR-грамматик.

См. также: книгу с драконом (Ахо, Сети, Ульман «Компиляторы. Принципы, Технологии, Инструменты»).