Восходящие детерминированные методы разбора

Precedence parsing

В. С. Полозов

Кафедра системого программирования СПбГУ

Теория автоматов и формальных языков

Восходящий разбор

Детерминированные восходящие анализаторы – наиболее исследуемые методы анализа.
Два класса:

- Чистые. Парсеры с приоритетами и BC-анализаторы.
- LR анализаторы - с добавлением нисходящей компоненты.

Восходящий разбор

Напомним:

- Правосторонний вывод: раскрыть самый правый нетерминал в сентенциальной форме заменой его на одну из правых частей его правил. Терминальная цепочка получается последовательным правосторонним выводом пока не останется нетерминалов.
- Шаг при восходящем разборе: выделить последний шаг правостороннего вывода и откатить его свернув сегмент входа к нетерминалу из которого он выводится. Выделенный сегмент называется *основой* (*handle*). Начинаем со всей цепочки, выделяем в обратном порядке.
- Для построения эффективного восходящиего анализатора надо научиться эффективно выделять основу.

Правосторонний вывод и анализ

S	
E	
EQF	
EQ	a
E	+a
EQF	+a
EQ	a+a
E	-a+a
F	-a+a
	a-a+a

Правосторонний вывод
(Right-most production)

	a-a+a
F	-a+a
E	-a+a
EQ	a+a
EQF	+a
E	+a
EQ	a
EQF	
E	
S	

Правосторонний анализ
(Right-most reduction)

Выделение управляющей конструкции

Рассмотрим пример:

$$4 + 5 \times 6 + 8$$

При просмотре выделяем основу 5×6 и сворачиваем:

$$4 + \underline{5 \times 6} + 8$$

$$\underline{4 + 30} + 8$$

$$\underline{34 + 8}$$

$$42$$

Если присмотреться, то виден и сдвиг:

$$4 + 5 \times 6 \qquad\qquad + 8$$

Выделение управляющей конструкции

Рассмотрим пример:

$$4 + 5 \times 6 + 8$$

При просмотре выделяем основу 5×6 и сворачиваем:

$$4 + \underline{5 \times 6} + 8$$

$$\underline{4 + 30} + 8$$

$$\underline{34 + 8}$$

$$42$$

Если присмотреться, то виден и сдвиг:

$$4 + 5 \times 6 \qquad\qquad + 8$$

Грамматика арифметических выражений

Арифметические выражения соответствуют грамматике:

$$
\begin{aligned}
S_S &\to \# \; E \; \# \\
E &\to E + T \\
E &\to T \\
T &\to T \times F \\
T &\to F \\
F &\to n \\
F &\to (\; E \;)
\end{aligned}
$$

Арифметические выражения со всеми скобками

Рассмотрим для примера более простую грамматику:

$$\begin{aligned} S_S &\to \# \; E \; \# \\ E &\to (\; E \; + \; T \;) \\ E &\to T \\ T &\to (\; T \; \times \; F \;) \\ T &\to F \\ F &\to n \\ F &\to (\; E \;) \end{aligned}$$

И пример:

$$\#((4+(5\times 6))+8)\#$$

Выделение основы – поиск первой закрывающей скобки с соответствующей открывающей.

Парсер с приоритетами

К сожалению, не всегда в грамматиках есть маркеры начала и конца для всех нетривиальных правых частей... Попробуем расставить?

- $+n\times \to +(n\times$
- $\times n+ \to \times n)+$
- $+n+ \to$?, скорее $+n)+$
- $\#n \to \#(n$
- $n\# \to n)\#$

Для нашего примера получили:

$$\#(4+(5\times 6)+8)\#$$

Неплохо, но скобки не все.

Парсер с приоритетами

К сожалению, не всегда в грамматиках есть маркеры начала и конца для всех нетривиальных правых частей... Попробуем расставить?

- $+n\times \to +(n\times$
- $\times n+ \to \times n)+$
- $+n+ \to$?, скорее $+n)+$
- $\#n \to \#(n$
- $n\# \to n)\#$

Для нашего примера получили:

$$\#(4+(5\times 6)+8)\#$$

Неплохо, но скобки не все.

Генераторы скобок

Идея: введём генераторы вместо скобок.

- $\lessdot$ - генератор открывающих скобок.
- $\gtrdot$ - генератор закрывающих скобок.
- $\doteq$ - не-скобки.

И скорректируем преобразование:

- $+\times \to +\lessdot\times$
- $\times+ \to \times\gtrdot+$
- $++ \to +\gtrdot+$
- $\#\ldots \to \#\lessdot\ldots$
- $\ldots\# \to \ldots\gtrdot\#$

Таблица приоритетов

Дополним и запишем в виде таблицы (приоритетов):

	#	+	×	(	)
#	$\doteq$	$\lessdot$	$\lessdot$	$\lessdot$	
+	$\gtrdot$	$\gtrdot$	$\lessdot$	$\lessdot$	$\gtrdot$
×	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
(		$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$
)	$\gtrdot$	$\gtrdot$	$\gtrdot$		$\gtrdot$

Стек при анализе:

- “Важные” символы - операторы.
- “Не важные” символы - числа.
- Генераторы скобок.

Примеры: $\#4+5\times6+8\#$
и со скобками (для $\doteq$): $\#4\times(5+6)\#$

Операторные грамматики

Определение

КС грамматика называется *операторной*, если в ней нет правил вида $A \to \varepsilon$ и $A \to ...BC....$

Введем понятия: оператор, $FIRST_{OP}(A)$, $LAST_{OP}(A)$.

Вычисление $FIRST_{OP}(A)$:

1. Для правил вида $A \to x\alpha, x \in V_T$ положим x в $FIRST_{OP}(A)$
2. Для правил вида $A \to B\alpha, B \in V_N$ положим $FIRST_{OP}(A) \leftarrow FIRST_{OP}(A) \cup FIRST_{OP}(B)$
3. Повторить шаг 2, если были изменения.

$LAST_{OP}(A)$ - аналогично.

Операторные грамматики

Определение

КС грамматика называется *операторной*, если в ней нет правил вида $A \to \varepsilon$ и $A \to ...BC....$

Введем понятия: оператор, $FIRST_{OP}(A)$, $LAST_{OP}(A)$.
Вычисление $FIRST_{OP}(A)$:

1. Для правил вида $A \to x\alpha, x \in V_T$ положим x в $FIRST_{OP}(A)$
2. Для правил вида $A \to B\alpha, B \in V_N$ положим $FIRST_{OP}(A) \leftarrow FIRST_{OP}(A) \cup FIRST_{OP}(B)$
3. Повторить шаг 2, если были изменения.

$LAST_{OP}(A)$ - аналогично.

Вычисление таблицы приоритетов

Построим отношение:

- Для каждого вхождения $q_1 q_2$ или $q_1 A q_2$, установим $q_1 \doteq q_2$.
- Для каждого вхождения $q_1 A$, установим $q_1 \lessdot q_2, q2 \in FIRST_{OP}(A)$.
- Для каждого вхождения $A q_1$, установим $q_2 \gtrdot q_1, q2 \in LAST_{OP}(A)$.

Если нет конфликтов - то это таблица приоритетов.

Обсуждение

Обсуждение:

- Легко построить, даже вручную.
- Достаточно эффективно.
- Многие практические грамматики являются [почти-]приоритетными.
- Из-за игнорирования некоторых нетерминалов строится скелетное дерево.
- Принимаются некорректные входы.

Оптимизации

- Часто можно заменить таблицу на функцию приоритета (всегда можно на две).
- Simple-precedence parsing, Weak-precedence parsing
- Bounded-context parsing - BC(2,1)

Обсуждение

Обсуждение:

- Легко построить, даже вручную.
- Достаточно эффективно.
- Многие практические грамматики являются [почти-]приоритетными.
- Из-за игнорирования некоторых нетерминалов строится скелетное дерево.
- Принимаются некорректные входы.

Оптимизации

- Часто можно заменить таблицу на функцию приоритета (всегда можно на две).
- Simple-precedence parsing, Weak-precedence parsing
- Bounded-context parsing - BC(2,1)

Вопросы