

Восходящие детерминированные методы разбора

Precedence parsing

В. С. Полозов

Кафедра системного программирования СПбГУ



Теория автоматов и формальных языков

Восходящий разбор

Детерминированные восходящие анализаторы – наиболее исследуемые методы анализа.

Два класса:

- Чистые. Парсеры с приоритетами и ВС-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 восходящего 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)

	F		a-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:

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

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:

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

Грамматика арифметических выражений

Арифметические выражения соответствуют грамматике:

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

Арифметические выражения со всеми скобками

Рассмотрим для примера более простую грамматику:

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

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?

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

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?

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

Для нашего примера получили:

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

Неплохо, но скобки не все.

Генераторы скобок

Идея: введём генераторы вместо скобок.

- \triangleleft - генератор открывающих скобок.
- \triangleright - генератор закрывающих скобок.
- $\dot{=}$ - не-скобки.

И скорректируем преобразование:

- $+ \times \rightarrow + \triangleleft \times$
- $\times + \rightarrow \times \triangleright +$
- $++ \rightarrow + \triangleright +$
- $\# \dots \rightarrow \# \triangleleft \dots$
- $\dots \# \rightarrow \dots \triangleright \#$

Таблица приоритетов

Дополним и запишем в виде таблицы (приоритетов):

	#	+	×	()
#	$\dot{=}$	\leftarrow	\leftarrow	\leftarrow	
+	\succ	\succ	\leftarrow	\leftarrow	\succ
×	\succ	\succ	\succ	\leftarrow	\succ
(\leftarrow	\leftarrow	\leftarrow	$\dot{=}$
)	\succ	\succ	\succ		\succ

Стек при анализе:

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

Примеры: $\#4 + 5 \times 6 + 8\#$

и со скобками (для $\dot{=}$): $\#4 \times (5 + 6)\#$

Операторные грамматики

Определение

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

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

Вычисление $FIRST_{OP}(A)$:

- 1 Для правил вида $A \rightarrow x\alpha$, $x \in V_T$ положим x в $FIRST_{OP}(A)$
- 2 Для правил вида $A \rightarrow B\alpha$, $B \in V_N$ 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 $A \rightarrow \varepsilon$ и $A \rightarrow \dots BC \dots$

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

Вычисление $FIRST_{OP}(A)$:

- 1 Для правил вида $A \rightarrow x\alpha, x \in V_T$ положим x в $FIRST_{OP}(A)$
- 2 Для правил вида $A \rightarrow B\alpha, B \in V_N$ 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_1q_2 или q_1Aq_2 , установим $q_1 \dot{=} q_2$.
- Для каждого вхождения q_1A , установим $q_1 \lessdot q_2, q_2 \in FIRST_{OP}(A)$.
- Для каждого вхождения Aq_1 , установим $q_2 \gtrdot q_1, q_2 \in LAST_{OP}(A)$.

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