Parsing Expression Grammars

А. А. Рубцов

7 декабря 2021 г.

1 Введение

Мы переходим к изучению новго формализма описания формальных языков — Parsing Expression Grammars, на русский это название можно перевести как «грамматики, разбирающие выражения». Этот перевод, увы не отражает сути, поэтому мы будем называть их PEG. В результате распознавания слова PEG-грамматикой получается дерево вывода (дерево разбора), что схоже с КС-грамматикой и также удобно для практики, а именно для вычислений по дереву разбора — важной части процесса компиляции и исполнения программы; впрочем это полезно и для более простых, но важных задач обработки текстов.

Перед описанием синтаксиса предлагаем взглянуть на описание PEG и КС-грамматики и увидеть сходство в их форме, однако предостерёжем, что смысл схожих обозначений сильно отличается!

PEG	**CF-Grammar**
$S \leftarrow AB$	$S \to AB$
$A \leftarrow B \,/\, C$	$A \to B \mid C$
$D \leftarrow \mathbf{!}B$	

Даже конкатенация в PEG имеет не тот же смысл, к которому мы привыкли. Чтобы во всём разобраться мы сначала зафиксируем символы, которые встречаются при описании PEG:

A нетерминал (обозначается большой буквой, как правило из начала алфавита)

a терминал (обозначается маленькой буквой из начала алфавита)

$\leftarrow$ отделяет левую часть правила от правой: $A \leftarrow \alpha$

$/$ разделитель между правилами для одного нетерминала: $A \leftarrow \beta \,/\, \gamma$

$\cdot$ «конкатенация», точка часто опускается

! отрицание

Fail ошибка

(открывающая скобка

) закрывающая скобка

ε пустое слово

Существенные различия между PEG и КС-грамматиками начинаются с операции «конкатенации», которую мы далее будем упоминать без кавычек. В отличие от стандартного определения, она определена правоассациативно, т. е. $ABCD$ интерпретируется как $A(B(CD))$ и расстановка скобок в словах будет иметь существенное значение. Поэтому мы определяем сентенциальную форму не как слово в алфавите нетерминалов и терминалов $N \cup \Sigma$, а как слово в алфавите $N \cup \Sigma \cup \{(,), \mathbf{!}\}$, которое превращается в правильное скобочное выражение при стирании всех символов, кроме скобок. Мы обозначаем сентенциальные формы маленькими греческими буквами: $\alpha, \beta, \gamma, \ldots$.

PEG определяется набором (N, Σ, P, S), смысл компонент в котором тот же, что и в КС-грамматиках. Нетерминалы N и терминалы Σ — конечные множества. Правила в PEG выглядят схоже с правилами в КС-грамматиках: в левой части правил всегда ровно один нетерминал, а в правой части находятся сентенциальные формы, разделённые символом $/$. В отличие от КС-грамматик, порядок правил важен! $A \leftarrow \alpha \,/\, \beta$ отличается от $A \leftarrow \beta \,/\, \alpha$. Более того, все правила для одного нетерминала должны быть перечисленны через разделитель $/$, в P не может встретиться два разных правила вида $A \leftarrow \alpha_1 \,/\, \alpha_2 \,/ \ldots / \,\alpha_n$ и $A \leftarrow \beta_1 \,/\, \beta_2 \,/ \ldots / \,\beta_k$.

Назовём *выражением* последовательность сентенциальных форм $\alpha_1 \,/\, \alpha_2 \,/ \ldots / \,\alpha_n$, состоящее быть может из одного элемента. Назовём *конфигурацией* пару (e, u), где e — выражение, а $u \in \Sigma^*$ — слово.

Разбор входного слова получается в результате вычисления функции R, которая получает на вход конфигурацию и возвращающет либо слово (в алфавите Σ), либо ошибку **Fail**. Перед формальным определением R опишем общую идею работы PEG. От входного слова последовательно отрезаются префиксы,

пока не получится превратить его в пустое слово (в этом случае, слово принято) либо не случится ошибка (в этом случае слово отвергнуто). В случае, когда конфигурация является конкатенацией, происходит рекурсивный вызов для её первого аргумента; в случае возникновения ошибки, идёт попытка попробовать следующее правило после /. Если же ошибка возникла под отрицанием, результат ошибки интерпретируется как пустое слово.

Переходим к формальному определению функции R — она определяется рекурсивно:

- $R(\varepsilon, u) = u$
- Для терминалов функция определена по «правилу сокращения»:
  $R(a, au) = u$, $R(a, bu) = \mathbf{Fail}$ при $b \neq a$, $R(a, \varepsilon) = \mathbf{Fail}$
- Для каждого правила $A \leftarrow e$: $R(A, u) = R(e, u)$
- Для конкатенации: если $R(X, u) = \varepsilon$, $X \in N \cup \Sigma$, то $R(X\alpha, uv) = R(\alpha, v)$;
  если $R(X, uv) = \mathbf{Fail}$, то $R(X\alpha, uv) = \mathbf{Fail}$
- Упорядоченный выбор: если $R(\alpha, u) = \mathbf{Fail}$, то $R(\alpha \,/\, e, u) = R(e, u)$;
  если $R(\alpha, u) = v$, то $R(\alpha \,/\, e, u) = v$
- Отрицание: если $R(\alpha, u) = \mathbf{Fail}$, то $R(\mathbf{!}\alpha, u) = \varepsilon$;
  если $R(\alpha, u) = v$, то $R(\mathbf{!}\alpha, u) = \mathbf{Fail}$

Слово w порождается PEG G, если $R(S, w) = \varepsilon$. Дерево вывода восстанавливается из дерева рекурсивных вызовов функции R, ветки в которых случился **Fail** удаляются из дерева, таким образом альтернативный выбор / в дереве не фигурирует.

Перейдём к примерам.

Пример 1. PEG $S \leftarrow aS \,/\, bS \,/\, \varepsilon$ порождает любое слово в алфавите $\{a, b\}$. Приведём пример наивного разбора (вычисления напрямую по определению) слова ab.

Мы обозначаем через $\vdash$ рекурсивный вызов R, а через $\dashv$ выход из рекурсивного вызова. При сокращении шагов мы используем $\vdash^*$ и $\dashv^*$.

$$(S, ab) \vdash (aS \,/\, bS \,/\, \varepsilon, ab) \vdash (aS, ab) \vdash (a, ab) \dashv (S, b)$$
$$(S, b) \vdash (aS \,/\, bS \,/\, \varepsilon, b) \vdash (aS, b) \vdash (a, b) \vdash \mathbf{Fail} \dashv^* (bS \,/\, \varepsilon, b)$$
$$(bS \,/\, \varepsilon, b) \vdash (bS, b) \vdash (b, b) \dashv (S, \varepsilon)$$
$$(S, \varepsilon) \vdash (aS \,/\, bS \,/\, \varepsilon, \varepsilon) \vdash (aS, \varepsilon) \vdash^* \mathbf{Fail} \dashv^* (bS \,/\, \varepsilon, \varepsilon)$$
$$(bS \,/\, \varepsilon, \varepsilon) \vdash^* \mathbf{Fail} \vdash (\varepsilon, \varepsilon) \dashv^* \varepsilon$$

Как видно из этого простого примера, ручная демонстрация наивного разбора PEG слишком трудозатратна. В следующих примерах мы будем опускать значительную часть вычислений, а проверить себя можно с помощью онлайн-конструктора `https://pegjs.org/online`.

Пример 2. PEG $S \leftarrow aSb \,/\, \varepsilon$ порождает язык $a^n b^n$.

Упражнение. Докажите, что любую LL(1)-грамматику можно превратить в PEG тривиальной заменой | на / и $\to$ на $\leftarrow$; правила нужно упорядочить так, чтобы правило $A \to \alpha$ с $\varepsilon \in \mathrm{FIRST}(\alpha)$ шло последним, если оно есть, порядок остальных правил неважен.

Пример 3. $S \leftarrow A \,/\, B,\ A \leftarrow aAb \,/\, ab,\ B \leftarrow aBc \,/\, ac$ порождает язык $\{a^n b^n \mid n > 0\} \cup \{a^n c^n \mid n > 0\}$.

$$(S, a^n c^n) \vdash (A \,/\, B, a^n c^n) \overset{*}{\vdash} (A, a^n c^n) \overset{*}{\vdash} (Ab^n, c^n) \overset{*}{\vdash} \textbf{Fail} \dashv^* (B, a^n c^n)$$
$$(B, a^n c^n) \overset{*}{\vdash} \varepsilon$$

В следующих примерах мы разберём особенности оператора **!**.

Пример 4. PEG $G : S \leftarrow (!A)B,\ A \leftarrow aAb \,/\, ab,\ B \leftarrow aB \,/\, bB \,/\, \varepsilon$ не порождает дополнение языка $a^n b^n$ как могло бы показаться, поскольку

$$(S, aabbab) \vdash ((!A)B, aabbab) \vdash (!A, aabbab) \overset{*}{\vdash} (\varepsilon, ab) \vdash ab \dashv^* R(!A, aabbab) = \textbf{Fail} \dashv^* \textbf{Fail}$$

А значит слово $aabbab$ не порождается PEG G. Эту проблему на практике можно решить добавлением маркера конца слова.

Двойное применение оператора **!**, т. е. $!(!A)$, позволяет проверить, что префикс необработанной части слова порождается нетерминалом A. Это применение встречается часто, поэтому для него используют специальное обозначение: $\&A = !(!A)$.

Пример 5. PEG $S \leftarrow (\&(Ac))BC,\ A \leftarrow aAb \,/\, ab,\ B \leftarrow aB \,/\, \varepsilon,\ C \leftarrow bCc \,/\, bc$ порождает язык $\{a^n b^n c^n \mid n > 0\}$.

Упражнение.** Придумайте алгоритм, который парсит PEG за линейное время