Алгоритм Эрли.

Алексей Сорокин

1 Введение.

Алгоритм Эрли представляет собой нисходящий алгоритм синтаксического разбора, то есть построение дерева разбора осуществляется сверху вниз. Хотя, как и в алгоритме Кока-Янгера-Касами, верхняя оценка на временную сложность алгоритма является кубической по длине слова, на практике константа в алгоритме Эрли значительно ниже. Кроме того, для однозначных грамматик доказана квадратичная верхняя оценка на время работы алгоритма, а для многих грамматик в реальности сложность оказывается линейной. В силу относительной простоты алгоритма это позволяет использовать его в отдельных практических приложениях (хотя в целом LR- и GLR-алгоритмы разбора являются значительно более популярными).

2 Краткое описание алгоритма.

Алгоритм Эрли получает на вход контекстно-свободную грамматику $G = \langle \Sigma, N, P, S \rangle$ и проверяет выводимость слова w в грамматике G. Для кодирования выводов в данной грамматике используются так называемые *ситуации*, имеющие вид $(A \to \alpha \cdot \beta, i)$, где $(A \to \alpha\beta) \in P$, $\cdot$ — вспомогательный символ, не принадлежащий $\Sigma \cup N$, $i \in \overline{0, |w|}$. Ситуации хранятся в множествах $D_0, \ldots, D_{|w|}$, причём наличие ситуации $(A \to \alpha{\cdot}\beta, i)$ в множестве D_j равносильна выполнению следующих условий (через w_{kl} обозначено подслово слова w с k-ой по l-ую позицию, причём позиции нумеруются с 0):

$$\exists \gamma \in (\Sigma \cup N)^* ((S \vdash w_{0i} A \gamma) \wedge A \vdash w_{ij})$$

При этом можно считать, что рассматриваются только левосторонние выводы в грамматике G. Для удобства предположим, что грамматика G содержит правило $S \to S_1$, причём S не входит в другие правила грамматики. Тогда выводимость w в грамматике G равносильна условию $(S \to S_1 \cdot, 0) \in D_{|w|}$. Алгоритм Эрли рекурсивно строит множества $D_0, \ldots, D_{|w|}$, поддерживая сформулированный выше инвариант.

3 Псевдокод.

Исходные параметры: Контекстно-свободная грамматика G, слово $w \in \Sigma^*$.
Результат: **True**, если $w \in L(G)$, **False**, иначе.
Применить алгоритм 2 к входным данным.
if $(S \to S_1 \cdot, 0) \in D_{|w|}$ **then**
  return ***True***
else
  return ***False***
end

Алгоритм 1: Проверка выводимости слова w в грамматике G.

Исходные параметры: Контекстно-свободная грамматика G, слово $w \in \Sigma^*$.
Результат: Множества ситуаций $D_0, \ldots, D_{|w|}$.
Инициализация:
$D_0 \leftarrow \{(S \to \cdot S_1, 0)\}$
for $i = 1, \ldots, |w|$ **do**
  $D_i \leftarrow \emptyset$;
end
Шаг работы:
for $j = 0, \ldots, |w|$ **do**
  $Scan(D, j)$;
  while D_j *изменяется* **do**
    $Complete(D, j)$;
    $Predict(D, j)$;
  end
end
Function *Scan(D, j)*
  if $j = 0$ **then**
    return
  end
  for $(A \to \alpha \cdot a\beta, i) \in D_{j-1}$ **do**
    if $a = w[(j-1)]$ **then**
      D_j.add $((A \to \alpha a \cdot \beta, i))$
    end
  end
end
Function *Predict(D,j)*
  for $(A \to \alpha \cdot B\beta, i) \in D_j$ **do**
    for $(B \to \gamma) \in P$ **do**
      D_j.add $(B \to \cdot\gamma, j)$
    end
  end
end
Function *Complete(D,j)*
  for $(B \to \gamma\cdot, i) \in D_j$ **do**
    for $(A \to \alpha \cdot B\beta, k) \in D_i$ **do**
      D_j.add $(A \to \alpha B \cdot \beta, k)$
    end
  end
end

Алгоритм 2: Алгоритм построения множеств $D_0, \ldots, D_{|w|}$.

4 Доказательство корректности.

Нужно доказать, что алгоритм 2 правильно строит множества $D_0, \ldots, D_{|w|}$, то есть что он поддерживает инвариант

$$(A \to \alpha \cdot \beta, i) \in D_j \leftrightarrow \exists \delta \in (\Sigma \cup N)^*((S \vdash w_{0i}A\delta) \wedge A \vdash w_{ij}).$$

Докажем импликацию слева направо индукцией по построению множеств D_j. Для этого нужно разобрать, в результате применения какой из инструкций алгоритма ситуация $(A \to \alpha \cdot \beta, i)$ попадает в множество D_j. База индукции, ситуация $(S \to S_1, 0) \in D_0$, очевидным образом удовлетворяет инварианту. Докажем шаг индукции.

Пусть ситуация $(A \to \alpha \cdot \beta, i)$ попала в D_j в результате применения правила *Scan*. В этом случае имеем $\alpha = \alpha' a$, $a = w[j-1]$ и $(A \to \alpha' \cdot a\beta, i) \in D_{j-1}$. По предположению индукции имеем $S \vdash w_{0i}A\delta$, что нам и было нужно, и $\alpha' \vdash w_{i(j-1)}$, тогда в силу $a = w[j-1]$ получаем $\alpha = \alpha' a \vdash w_{i(j-1)}w[j-1] = w_{ij}$, что и требовалось.

Теперь пусть ситуация $(A \to \alpha \cdot \beta, i)$ попала в D_j в результате применения правила *Predict*. По построению получаем, что $\alpha = \varepsilon$, $i = j$, что автоматически влечёт второй пункт утверждения. Кроме того, найдутся $i' \leqslant i$ и ситуация $(A' \to \alpha' \cdot A\delta', i') \in D_i$, откуда по предположению индукции имеем $S \vdash w_{0i'}A'\delta''$, $\alpha' \vdash w_{i'i}$. Получаем $S \vdash w_{0i'}A'\delta'' \vdash_1 w_{0i'}\alpha' A\delta'\delta'' \vdash w_{0i'}w_{i'i}A\delta'\delta'' = w_{0i}A\delta$, что и требовалось.

Осталось разобрать случай, когда ситуация $(A \to \alpha \cdot \beta, i)$ попала в D_j в результате применения правила *Complete*. По построению $\alpha = \alpha' A'$ и найдутся i' и δ, такие что $(A \to \alpha' \cdot A'\beta, i) \in D_{i'}$ и $(A' \to \gamma \cdot, i') \in D_j$. Следовательно, $\alpha = \alpha' A' \vdash w_{ii'}w_{i'j} = w_{ij}$, что и было нужно. Кроме того, $S \vdash w_{0i}A\delta$ по предположению индукции, что и требовалось доказать.

В одну сторону равносильность доказана, докажем в противоположную. Доказательство проведём индукцией по суммарной длине вывода $w_{0i}A\delta$ из S и w_{ij} из α, после чего применим индукцию по длине вывода w_{ij} из α. Разберём несколько случаев в зависимости от последнего символа α. Если $\alpha = \alpha' a$, тогда $a = w[j-1]$, $\alpha' \vdash w_{i(j-1)}$. По предположению индукции получаем $(A \to \alpha' \cdot a\beta, i) \in D_{j-1}$. Тогда по правилу *Scan* получаем $(A \to \alpha' a \cdot \beta, i) \in D_j$, что и требовалось.

Если $\alpha = \alpha' B$, тогда получаем, что существует i', такой что $\alpha' \vdash w_{ii'}$, $B \vdash w_{i'j}$. Тогда имеем $(A \to \alpha' \cdot B\beta) \in D_{i'}$. Кроме того, можно записать $S \vdash w_{0i}A\delta \vdash w_{0i}w_{ii'}B\beta\delta$, а также $B \vdash_1 \gamma \vdash w_{i'j}$. Применяя индукцию по второму параметру, имеем $(B \to \gamma \cdot, i') \in D_j$, откуда по правилу *Complete* получаем $(A \to \alpha' B \cdot \beta) \in D_j$, что и требовалось.

Пусть теперь $\alpha = \epsilon$, тогда $i = j$. Тогда либо $i = 0$, $A = S_1$, $\delta = \epsilon$, что доказывает базу индукции, либо вывод можно переписать в виде $S \vdash w_{0i'}A'\delta'' \vdash_1 w_{0i'}w_{i'i}A\delta'\delta'' = w_{0i}A\delta$ для некоторого правила $(A' \to w_{i'i}A\delta') \in P$. Отсюда по предположению индукции следует, что $(A' \to \cdot w_{i'i}A\delta', i') \in D_{i'}$, что после нескольких применений правила *Scan* приводит к $(A' \to w_{i'i} \cdot A\delta', i') \in D_i$, после чего по правилу *Predict* мы получаем $(A \to \cdot \beta, i) \in D_i$, что и требовалось. Корректность доказана.

5 Анализ сложности алгоритма.

Обозначим через $|G|$ суммарную длину всех правил из множества P (очевидно, что при фиксированном стартовом нетерминале правил достаточно для задания грамматики). Для хранения всех ситуаций из каждого из множеств D_j тогда требуется не больше чем $O(|G||w|)$ памяти, что в сумме даёт пространственную сложность $O(||G||w|^2)$. Для оптимизации временных затрат для каждой ситуации $(A \to \alpha \cdot \beta, i) \in D_j$ нужно хранить ссылку на ситуации вида $(A' \to \alpha' \cdot A\beta') \in D_i$, с которыми данная ситуация может «взаимодействовать» при применении инструкции $Complete$. Для этого достаточно либо поддерживать соответствующие обратные ссылки при появлении новых ситуаций, либо просто обеспечить быстрый доступ ко всем ситуациям со вторым индексом i в множестве D_j. Второе легко достигается при хранении множества D_j с помощью булева массива или массива множеств, индексированного возможными вторыми индексами.

Рассмотрим временную сложность при хранении D_j в виде булева массива. Тогда на каждой итерации алгоритма на процедуру $Scan$ тратится $O(|D_{j-1}|)$, а на процедуру $Predict$ — $O(|D_j|)$ времени, что даёт затраты $O(|G||w|)$ на каждой итерации. При операции $Complete$ максимально возможное затраченное время на каждую ситуацию $(A \to \alpha \cdot \beta, i) \in D_j$ равно $O(|D_i|)$, что в сумме приводит к максимальным затратам на итерации на процедуру $Complete$, равным $O(|G|^2|w|^2)$. Суммарные затраты времени, таким образом, равны $O(|G|^2|w|^2)$ на одну итерацию и $O(|G|^2|w|^3)$ на весь алгоритм.

Данная оценка является сильно огрублённой, поскольку в большинстве случаев множества D_j оказываются существенно меньше. Кроме того, если для грамматики удаётся доказать, что число появлений каждой ситуации ограничено некоторым числом C, то суммарная сложность при правильной реализации не превысит C на число ситуаций, то есть будет квадратичной.