Алгоритм LR определения принадлежности слова языку, задаваемому контекстно-свободной грамматикой

Обозначения:

Г - сумма длин всех правил

Описание основных функций:

1. $transitive\_closure(initial\_state, f)$

   Эта функция берет начальное состояние некой структуры $initial\_state$, и применяет к нему функцию f до тех пор, пока $inititial\_state$ не перестанет меняться под ее действием - а именно, она останавливается в первый момент, когда $state$ и $f(state)$ равны, потому что тогда, по индукции, любое следующее применение f не будет менять структуру. Асимптотика очень сильно зависит от специфики f, поэтому ее можно описать только как $O(f\_max\_iterations \cdot T(f))$.

2. $build\_first\_table()$

   Эта функция строит First-таблицу. Для этого она хранит таблицу, в которой для каждого нетерминала хранится структура $DecayedCell$, в которой отдельно хранятся наборы нетерминалов и терминалов - они отвечают за первые нетерминалы и терминалы, соответственно, которые могут быть выведены из данного нетерминала. В начальный момент времени для каждого нетерминала туда кладутся только первые символы правил. На очередном шаге для каждого нетерминала H:

   1. Для каждого нетерминала A из набора нетерминалов в $DecayedCell[H]$ положим в список нетерминалов $DecayedCell[H]$ все нетерминалы из $DecayedCell[A]$, а также в список терминалов - все терминалы из $DecayedCell[A]$.
   2. Рассмотрим все правила в грамматике, в которых в левой части стоит H. Для каждого правила перебираем символы с некой метки, о которой будет сказано потом. Если текущий символ - нетерминал, то мы добавляем его в список нетерминалов $DecayedCell[H]$ - более того, если он не ϵ-порождающий в данный момент (то есть не содержит пустую строку в своем $DecayedCell$), то на этом мы заканчиваем рассматривать правило, и ставим метку перед этим нетерминалом, иначе переходим к следующему символу в правиле. Если это терминал, то мы просто добавляем его в список терминалов $DecayedCell[H]$ и ставим метку перед терминалом. Таким образом, метка обозначает, где нам продолжить в следующий раз, потому что если мы уже дошли до данной метки, то все нетерминалы до этого момента могли раскрыться в пустое слово, поэтому их не надо снова добавлять.

   Шаги осуществляются с помощью транзитивного замыкания. Таким образом, каждая итерация без учета прохода по грамматике работает за $O(Table\_size)$, где $Table\_size$ - текущий размер таблицы, а он ограничен $O(|N| \cdot (|N| + |\Sigma|))$. На каждой итерации добавляется хотя бы один символ, поэтому итераций не больше $O(|N| \cdot (|N| + |\Sigma|))$, при этом грамматику мы обходим суммарно один раз из-за меток, поэтому итоговая асимптотика построения таблицы $O(\Gamma + |N|^2(|N| + |\Sigma|)^2)$.

3. $state\_closure(kernel)$

   Эта функция берет набор состояний $kernel$, и делает его замыкание, то есть раскрывает все нетерминалы, перед которыми стоит точка в состоянии. Это тоже делается с помощью транзитивного замыкания и таблицы $First$ простым алгоритмом, детали которого можно найти непосредственно в коде. Оценим количество всевозможных состояний. Точка может находиться в Γ позициях, всего правил может быть $|P|$, а также в состоянии мы храним $lookup$ - символ, который должен быть в следующим в слове, если мы хотим свернуться по этому правилу при парсинге. $lookup$'ы оцениваются количеством терминалов, поэтому всего состояний может быть $\Gamma \cdot |P| \cdot |\Sigma|$ - это не асимптотика, просто оценил количество состояний, что понадобится в дальнейшем.

4. $build\_automata()$

   Эта функция собственно строит автомат. Для этого хранится очередь из ядер, по которым нужно построить состояния. В начальный момент там хранится единственное правило, в котором в левой части стоит $initial\_symbol$ (его единственность проверяется в самом первом вызываемом методе - $check\_grammar$). Далее для каждого очередного ядра состояний делается следующее:

   1. Составить замыкание состояния с помощью $state\_closure$
   2. Для каждого нетерминала и терминала надо провести ребро по этому символу, и посмотреть, какие состояния будут составлять ядро состояния, в которое делается переход, и добавить это ядро в очередь.

   Так, суммарно асимптотика всех $state\_closure$ оценивается количеством возможных состояний, при этом суммарное число проведенных ребер оценивается числом правил, потому что каждый узел содержит хотя бы одно правило. Таким образом, итоговая асимптотика будет $O(\Gamma \cdot |P| \cdot |\Sigma| + (|N| + |\Sigma|) \cdot |P|) = O(\Gamma \cdot |P| \cdot |\Sigma|)$.

5. $parse(word)$

   Этот метод определяет, лежит ли слово в языке. Для этого создаются два стека. Первый обозначает путь из номеров узлов, который мы прошли, чтобы дойти до текущего узла, второй - символы, которые мы прочитали и свернули по каким-то правилам. На каждом шаге мы либо проходим по следующему символу в слове по ребру, либо сворачиваемся по какому-то правилу (детерминировано, если грамматика является $lr(1)$). Сворачивание оценивается количеством пройденных состояний, потому что при сворачивании мы идем обратно по ребрам. А продвигаемся в следующее состояние мы по прочитанным буквам либо по одному нетерминалу, поэтому в среднем на каждую букву в слове приходится $O(1)$ возвратов по ребрам. Проход по букве - чистое $O(1)$, поэтому сам $parse$ работает за $O(|word|)$.

Таким образом, асимптотики:

1. Предподсчет: $O(\Gamma|P||\Sigma| + |N|^2(|N| + |\Sigma|)^2)$
2. Парсинг каждого слова длины n: $O(n)$.