Кодирующие КС-языки
Нисходящий разбор

Теория формальных языков

2022 г.

Кодировка путей праволинейной грамматики

Рассмотрим путь вывода произвольного слова $a_1 \dots a_n$ в праволинейной грамматике. Он имеет вид $S \to a_1 A_1; A_1 \to a_2 A_2; \dots A_n \to a_n$. Применим к нему обратный гомоморфизм $h(A_i; A_i \to) = \varepsilon$ и сотрём префикс $S \to$, получим искомое слово.

Алфавит: $\Sigma \cup N \cup \{;, \to\}$. Описание языка: $\{S \to a_i (A_i; A_i \to a_j)^*\}$.

Описание языка привязано к множеству нетерминалов в рассматриваемой грамматике, но левые и правые вхождения нетерминалов можно было бы закодировать скобками, по количеству считающими номер нетерминала.

Кодировка путей КС-грамматики

Аналогично — с кодировкой путей вывода в КС-грамматике. Но здесь есть проблема с "перепутанными скобками": в пути $A_1 \to a_1 A_2 A_3; A_2 \to \ldots; A_3 \to \ldots$ имена A_2 и A_3 состоят в зависимости с соответствующими левыми частями, но перемежаются.

Решение

Поменять местами кодировки для A_2 и A_3 в правых частях.

Язык Грейбах

Здесь ε-free вариант. D — язык сбалансированных скобочных структур над $\{(,), [,]\}$.

$L_0 = \{x_1 c y_1 c z_1 d \dots d x_n c y_n c z_n d \mid y_1 \dots y_n \in eD$ & z_i, x_i не содержат e & $y_1 \in e\{(,), [,]\}^*$ & $y_{i+1} \in \{(,), [,]\}^*\}$

Язык Грейбах

Здесь ε-free вариант. D — язык сбалансированных скобочных структур над $\{(,),[,]\}$.

$L_0 = \{x_1 c y_1 c z_1 d \dots d x_n c y_n c z_n d \mid y_1 \dots y_n \in eD$ & z_i, x_i не содержат e & $y_1 \in e\{(,),[,]\}^*$ & $y_{i+1} \in \{(,),[,]\}^*\}$

Утверждение

Если L — КС-язык, тогда существует $h \in \mathrm{Hom}$ такой, что $h^{-1}(L_0) = L$.

Гомоморфизм Грейбах

Пусть G — грамматика для L в форме Грейбах (Шейлы!). Пронумеруем нетерминалы G так, чтобы стартовый был первым. Построим вспомогательную функцию ξ:

- для правил $A_i \to a$ положим $\xi(i) =)]^{i})$
- для правил $A_i \to aA_{j1} \ldots A_{jn}$ положим $\xi(i) =)]^{i})([^{jm}(\ldots([^{j1}($
- если $i = 1$, тогда дополнительно припишем префикс $e([[($.

Пусть терминалом a начинаются левые части правил $k_1, \ldots, k_m$. Тогда $h(a) = c\xi(k_1)c \ldots c\xi(k_m)d$.

Коммутативный образ по Пиллингу

Лемма Ардена для коммутативных образов

Пусть правила грамматики имеют следующий вид:

$$T \to W_1 T^{i_1} T \mid \ldots \mid W_{k-1} T^{i_{k-1}} T \mid W_k$$

где W_i не содержит T, и степени i_j различны. При этом W_i могут содержать любые регулярные операции над константами и нетерминалами, отличными от T. Коммутативный образ правил для T есть $T = (W_1 (W_k)^{i_1})^* W_k + \dots + (W_{k-1} (W_k)^{i_{k-1}})^* W_k$.

Ограничение: меняет местами подвыводы, соединяя те, которые накачиваются отдельно.

μ-регулярные выражения и РБНФ

Определим оператор минимальной неподвижной точки: скажем, что $L(\mu X.r(X))$ — это объединение языков $r(X)$, $r(r(X))$, $\ldots$, $r^n(X)$.

Что будет, если добавить в регулярные выражения μ-оператор?

μ-регулярные выражения и РБНФ

Определим оператор минимальной неподвижной точки: скажем, что $L(\mu X.r(X))$ — это объединение языков $r(X)$, $r(r(X))$, $\dots$, $r^n(X)$.

Что будет, если добавить в регулярные выражения μ-оператор?

μ-оператор по переменным + регулярные операции определяют множество всех КС-языков.

Пример: $\mu y.a(\mu x.axb + y + a)$ определяет грамматику $Y \to a\,X$, $X \to a\,X\,b\,|\,Y\,|\,a$.

μ-регулярные выражения и РБНФ

Определим оператор минимальной неподвижной точки: скажем, что $L(\mu X.r(X))$ — это объединение языков $r(X), r(r(X)), \ldots,$ $r^n(X)$.

Что будет, если добавить в регулярные выражения μ-оператор?

μ-оператор по переменным + регулярные операции определяют множество всех КС-языков.

Пример: $\mu y.a(\mu x.axb + y + a)$ определяет грамматику $Y \to a\,X$, $X \to a\,X\,b\,|\,Y\,|\,a$.
Неудобно, если язык некоторого нетерминала используется в нескольких правилах (например, $S \to A\,B\,|\,B\,A$ заставит дважды выписывать μ-выражения для A и B).
В действительности аналогом μ-регулярных выражений выступает РБНФ: в правых частях правил разрешены любые регулярные операции.

Проблемы PDA ~~и сила Рефала~~

Не всегда по стеку и входным символам можно понять, что делать со стеком. Пример: грамматика палиндромов.

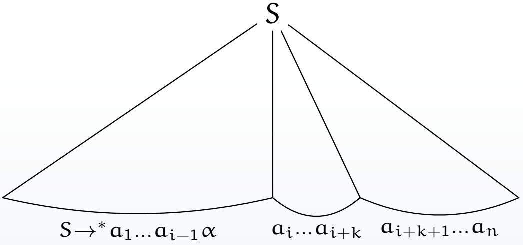
Неформально: заглядывание вперёд на произвольное, но заранее ограниченное число символов не даёт никакой информации о том, что делать со стеком.

- Возьмём слова $a^{2p}\,b\,a^{2p}$ и a^{2p}.
- После чтения p символов в первом случае нужно продолжать накапливать стек, а во втором — начинать вынимать из него.
- Но впереди одни и те же a^p букв...

Нисходящий разбор

Пусть PDA-анализатор хранит пару $\langle \alpha, a_i \dots a_{i+k} \rangle$, где α — сент. форма, $a_i \dots a_{i+k}$ — k следующих символов в строке.

- Если $\alpha = A\alpha'$, тогда по правилу $A \to \beta$ стековый анализатор переходит в $\langle \beta\alpha', a_i \dots a_{i+k} \rangle$, либо сообщает об ошибке.
- Если $\alpha = a_i\alpha'$, тогда a_i одновременно снимается со стека и читается во входной строке.



LL(k)-грамматики

Определение

Грамматика G называется LL(k) *(left-to-right, leftmost derivation)* $\Leftrightarrow$ в ситуации, когда существуют выводы
$S \to^* w_1 A \alpha \to w_1 \xi \alpha \to^* w_1 c w_2$,
$S \to^* w_1 A \alpha \to w_1 \eta \alpha \to^* w_1 c w_3$, причём $c \in \Sigma^k$, $w_1, w_2, w_3 \in \Sigma^*$, или $c \in \Sigma^{<k}$, $w_2 = w_3 = \varepsilon$, всегда $\eta = \xi$.

LL(k)-грамматики

Определение

Грамматика G называется LL(k) *(left-to-right, leftmost derivation)* $\Leftrightarrow$ в ситуации, когда существуют выводы
$S \to^* w_1 A\alpha \to w_1\xi\alpha \to^* w_1 c w_2$,
$S \to^* w_1 A\alpha \to w_1\eta\alpha \to^* w_1 c w_3$, причём $c \in \Sigma^k$, $w_1, w_2, w_3 \in \Sigma^*$, или $c \in \Sigma^{<k}$, $w_2 = w_3 = \varepsilon$, всегда $\eta = \xi$.

Неформально: неоднозначность в выборе правила грамматики при разборе сверху вниз устраняется заглядыванием вперёд на k букв.

Критерий LL(k)-грамматики

Определим

- $\mathrm{FIRST}_k(\eta) = \{a_1 \dots a_j \mid (j < k \;\&\; \eta \to a_1 \dots a_j) \vee (j = k \;\&\; \eta \to a_1 \dots a_k \alpha)\} \cup \{\varepsilon \mid \eta \to^* \varepsilon\}$
- $\mathrm{FOLLOW}_k(\eta) = \{a_1 \dots a_j \mid (j < k \;\&\; S \to^* \beta\eta a_1 \dots a_j) \vee (j = k \;\&\; S \to^* \beta\eta a_1 \dots a_k \alpha)\} \cup \{\$ \mid S \to^* \beta\eta\}$

Тогда G — LL(k)-грамматика $\Leftrightarrow \forall A \to \alpha, A \to \beta$ $(\mathrm{FIRST}_k(\alpha\, \mathrm{FOLLOW}_k(A)) \cap \mathrm{FIRST}_k(\beta\, \mathrm{FOLLOW}_k(A)) = \varnothing)$.

Вычисление FIRST_k

- $\mathrm{FIRST}_k(A) = \{a_1 \dots a_k \,|\, A \to a_1 \dots a_k \alpha\}$ $\cup \{a_1 \dots a_j \,|\, j < k \;\&\; A \to a_1 \dots a_j\}$;
- До исчерпания: $\forall A \to B_1 \dots B_n$, $\mathrm{FIRST}_k(A) = \mathrm{FIRST}_k(A) \cup \mathtt{first}_k(\mathrm{FIRST}_k(B_1) \dots \mathrm{FIRST}_k(B_n))$, где $\mathtt{first}_k$ — это k первых символов строки.

Вычисление FIRST_k

- $\mathrm{FIRST}_k(A) = \{a_1 \dots a_k \mid A \to a_1 \dots a_k \alpha\} \cup \{a_1 \dots a_j \mid j < k \;\&\; A \to a_1 \dots a_j\}$;
- До исчерпания: $\forall A \to B_1 \dots B_n$, $\mathrm{FIRST}_k(A) = \mathrm{FIRST}_k(A) \cup \mathtt{first}_k(\mathrm{FIRST}_k(B_1) \dots \mathrm{FIRST}_k(B_n))$, где $\mathtt{first}_k$ — это k первых символов строки.

Алгоритм задаёт фундированный порядок на множестве конфигураций $\mathrm{FIRST}_k(A_i)$, поэтому рано или поздно множества FIRST_k перестанут изменяться.

Вычисление FOLLOW_k

Добавляем правило из нового стартового символа $S_0 \to S\$$.

- $\mathrm{FOLLOW}_k(A) = \varnothing$ для всех $A \neq S$.
- До исчерпания: $\forall B \to \beta$ и разбиений $\beta = \eta_1 A \eta_2$, $\mathrm{FOLLOW}_k(A) =$ $\mathrm{FOLLOW}_k(A) \cup \mathtt{first}_k(\mathrm{FIRST}_k(\eta_2)\mathrm{FOLLOW}_k(B))$, где $\mathtt{first}_k$ — это k первых символов строки.

Алгоритм также задаёт wfo на множестве конфигураций $\mathrm{FOLLOW}_k(A_i)$.

Таблица LL(k)-разбора

Таблица $T_k(A, x)$ по нетерминалу A и lookahead-символам x вычисляет правило для парсинга.

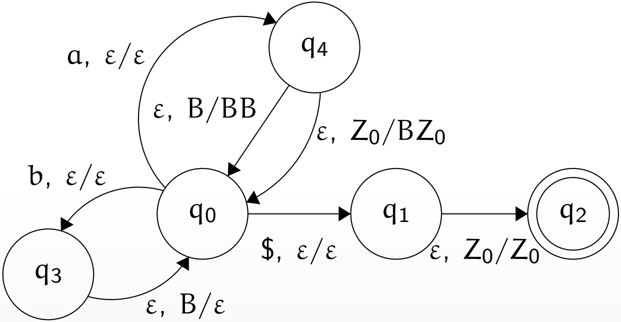
for all $A \to \alpha$
  for all $x \in \mathrm{first}_k(\mathrm{FIRST}_k(\alpha)\,\mathrm{FOLLOW}_k(A))$
    if $T_k(A, x)$ не задано, тогда $T_k(A, x) = A \to \alpha$
      else объявление о конфликте

PDA для LL(1)-грамматик

- Чтение терминалов с ленты происходит только в одном состоянии, при этом не меняется стек, но делается переход в другое состояние (учёт lookahead-ов).
- Во всех остальных состояниях только меняется стек. Переход ничего не читает с ленты и возвращает автомат в читающее состояние.

$$
\begin{array}{lcl}
S & \rightarrow & aBS \\
S & \rightarrow & \varepsilon \\
B & \rightarrow & aBB \\
B & \rightarrow & b
\end{array}
$$

$S \longrightarrow Z_0$ в PDA («дно»).



Для LL(k) аналогично, но состояния будут соответствовать k последним прочитанным буквам.

LL(k)-грамматики без ε-правил

Пример

Стандартный алгоритм удаления ε-правил приводит к разрушению LL-свойств:

$$\begin{array}{rcl} S & \rightarrow & ABC \\ A & \rightarrow & aA \mid d \\ B & \rightarrow & b \mid \varepsilon \\ C & \rightarrow & c \mid \varepsilon \end{array}$$

LL(k)-грамматики без ε-правил

Утверждение (Куроки–Суонио)

Всякая LL(k)-грамматика может быть преобразована в LL(k+1)-грамматику без ε-правил.

LL(k)-грамматики без ε-правил

Утверждение (Куроки–Суонио)

Всякая LL(k)-грамматика может быть преобразована в LL(k+1)-грамматику без ε-правил.

Идея доказательства

Сначала избавимся от всех правил, начинающихся с nullable-нетерминала, путём введения notnull-двойников.

Затем присоединим все nullable-нетерминальные отрезки, стоящие в правых частях, к предшествующим им notnull-нетерминалам, и объявим полученные строки новыми нетерминалами.

В новых правилах nullable-кусок приписывается к самому последнему нетерминалу в правой части.

Пример устранения ε без разрушения LL-свойства

$$
\begin{array}{ll}
S \to ABC \,|\, Bk & A \to aA \,|\, d \\
B \to b \,|\, \varepsilon & C \to c \,|\, \varepsilon
\end{array}
$$

Сначала избавимся от обнуляемого нетерминала в начале правой части правила стандартным приёмом:

$$
\begin{array}{lll}
S \to ABC \,|\, B'k \,|\, k & A \to aA \,|\, d & \\
B \to b \,|\, \varepsilon & C \to c \,|\, \varepsilon & B' \to b
\end{array}
$$

Пример устранения ε без разрушения LL-свойства

$$S \to ABC \mid B'k \mid k \quad A \to aA \mid d$$
$$B \to b \mid \varepsilon \quad C \to c \mid \varepsilon \quad B' \to b$$

Теперь присоединим правый обнуляемый контекст к A и протащим его по правилу переписывания для A:

$$S \to [ABC] \mid B'k \mid k \quad [ABC] \to a[ABC] \mid [dBC]$$
$$B \to b \mid \varepsilon \quad C \to c \mid \varepsilon \quad B' \to b$$

Пример устранения ε без разрушения LL-свойства

$$
\begin{array}{ll}
S \to [ABC] \mid B'k \mid k & [ABC] \to a[ABC] \mid [dBC] \\
B \to b \mid \varepsilon & C \to c \mid \varepsilon \quad B' \to b
\end{array}
$$

Определяем правила переписывания для нетерминала $[dBC]$, соответствующие коллапсу всего обнуляемого контекста, его префиксов, и непустой развёртке префикса.

$$
\begin{array}{ll}
S \to [ABC] \mid B'k \mid k & [ABC] \to a[ABC] \mid [dBC] \\
[dBC] \to d[bC] \mid dc \mid d & B \to b \mid \varepsilon \\
C \to c \mid \varepsilon & B' \to b
\end{array}
$$

Пример устранения ε без разрушения LL-свойства

$$
\begin{array}{ll}
S \to [ABC] \,|\, B'k \,|\, k & [ABC] \to a[ABC] \,|\, [dBC] \\
[dBC] \to d[bC] \,|\, dc \,|\, d & B \to b \,|\, \varepsilon \\
C \to c \,|\, \varepsilon & B' \to b
\end{array}
$$

Аналогично обрабатываем нетерминал $[bC]$ и удаляем все правила для обнуляемых нетерминалов из грамматики.

$$
\begin{array}{ll}
S \to [ABC] \,|\, B'k \,|\, k & [ABC] \to a[ABC] \,|\, [dBC] \\
[dBC] \to d[bC] \,|\, dc \,|\, d & [bC] \to bc \,|\, b \quad B' \to b
\end{array}
$$

Лемма Розенкранца–Стирнса

Утверждение

Если G — $LL(k)$-грамматика, тогда вышеописанный алгоритм устранения ε-правил всегда завершается, поскольку ни на одном шаге не появляется новых нетерминалов, дважды присоединяющих в правый контекст один и тот же обнуляемый нетерминал грамматики G.

Покажем, что в общем случае алгоритм может и не завершаться. Рассмотрим следующую грамматику, не являющуюся однозначной (и следовательно, $LL(k)$ ни для какого значения k).

$$S \to SS \mid a \mid \varepsilon$$

Лемма Розенкранца–Стирнса

Утверждение

Если G — $LL(k)$-грамматика, тогда вышеописанный алгоритм устранения ε-правил всегда завершается.

Покажем, что в общем случае алгоритм может и не завершаться. Рассмотрим следующую грамматику, не являющуюся однозначной (и следовательно, $LL(k)$ ни для какого значения k).

$$S \to SS \mid a \mid \varepsilon$$

По алгоритму, устраним сначала обнуляемый правый нетерминал.

$$S \to S'S \mid S' \mid a \mid \varepsilon \quad S' \to S'S \mid S' \mid a$$

Теперь присоединим обнуляемые контексты (это оставшиеся вхождения S) и посмотрим, какие правила получатся для нового нетерминала $[S'S]$.

$$S \to [S'S] \mid S' \mid a \mid \varepsilon \quad S' \to [S'S] \mid S' \mid a \quad [S'S] \to S'SS \mid S'S \mid aS$$

Лемма Розенкранца–Стирнса

Покажем, что в общем случае алгоритм может и не завершаться. Рассмотрим следующую грамматику, не являющуюся однозначной (и следовательно, $LL(k)$ ни для какого значения k).

$$S \to SS \mid a \mid \varepsilon$$

По алгоритму, устраним сначала обнуляемый правый нетерминал.

$$S \to S'S \mid S' \mid a \mid \varepsilon \qquad S' \to S'S \mid S' \mid a$$

Теперь присоединим обнуляемые контексты (это оставшиеся вхождения S) и посмотрим, какие правила получатся для нового нетерминала $[S'S]$.

$$S \to [S'S] \mid S' \mid a \mid \varepsilon \qquad S' \to [S'S] \mid S' \mid a \qquad [S'S] \to S'SS \mid S'S \mid aS$$

Видно, что нужно порождать новый нетерминал, потому что обнуляемый контекст у правила $[S'S] \to S'SS$ — это уже два вхождения S. После его порождения опять получим правило вида $[S'SS] \to S'SSS$, и вообще, для каждого нового $[S'S^n]$ — правило $[S'S^n] \to S'S^{n+1}$. Значит, ни на какой итерации процесс не сходится.

GNF для LL(k)-грамматики

Утверждения

- Ни одна леворекурсивная грамматика — не LL(k) ни для какого k.
- Всякая LL(k)-грамматика может быть преобразована в LL(k+1)-грамматику в GNF.

LL(k)-языки

Определение

CFL L — это LL(k)-язык, если для него существует LL(k)-грамматика.

LL(k)-языки

Определение

CFL L — это LL(k)-язык, если для него существует LL(k)-грамматика.

Существуют LL(k), но не LL(k-1)-языки.

Рассмотрим язык $\{a^n(b^k d \mid b \mid c)^n\}$. Это LL(k)-язык:

$$
\begin{array}{rcl}
S & \to & aCA \\
C & \to & aCA \mid \varepsilon \\
A & \to & bB \mid c \\
B & \to & b^{k-1}d \mid \varepsilon
\end{array}
$$

Неформально: не LL(k-1) потому, что иначе бы имелась LL(k)-грамматика в GNF для L. В стеке разбора для префикса a^n оказалось бы больше, чем $2k-1$ символов. Подставляя варианты суффиксов к a^{n+k-1}, получаем противоречие (метод подмены).

Метод подмены

Общая схема метода

- Рассматриваем сразу LL(k)-грамматику в GNF.
- Показываем, что в её стеке в состоянии α должно находиться не меньше, чем $f(k)$ разных символов при предъявлении тех же самых lookahead k символов.
- После данных k символов предъявляем длинный суффикс, отрезок которого должен контекстно-свободно выводиться из некоторого нетерминала в стеке.
- Предъявляем другой суффикс к тому же префиксу и lookahead, в котором не может быть такого отрезка, и подменяем вывод нетерминала.

Техника метода подмены

- Ищем такие два слова xfy, xfz, что $|f| = k$ и при чтении префикса x стек мог неограниченно разрастись (т.е. есть синхронная накачка хотя бы одной из пар x и y и x и z). Предполагаем высоту стека равной хотя бы $k + t$ (где t выбирается в зависимости от задачи).
- k символов в стеке при выталкивании точно распознают фрагмент f (lookahead). Рассматриваем, что может распознаться остальными t символами. Комбинируем распознанные фрагменты и показываем, что их комбинация не входит в язык.
- Либо если в стеке осталось t символов, а длина, например, y меньше t, тогда с этим стеком точно нельзя распознать y.

Пример подмены

Может ли язык $\{a^n b^n\} \cup \{a^n c^n\}$ задаваться LL(k)-грамматикой для некоторого k?

Пусть такое k нашлось (без ограничения общности — в GNF). Рассмотрим слово $a^{n+k}b^{n+k}$ и подберём такое n, что после чтения a^n LL-анализатор имеет в стеке не меньше $k+2$ символов. Пусть последний символ — это Y. Он распознает некоторое подслово суффикса b^{n+k}, а именно b^s. Теперь подменим строку на $a^{n+k}c^{n+k}$. В этом случае Y должен распознать некоторое c^t. Но значит, $a^{n+k}b^{n+k-s}c^t \in L$, что невозможно. $\square$

Опциональный Else

Рассмотрим GNF-грамматику для языка $\{a^n b^m \mid n \geqslant m\}$.

Опциональный Else

Рассмотрим GNF-грамматику для языка $\{a^n b^m \mid n \geqslant m\}$. Положим $w = a^{n+k} b^{n+k}$, где n таково, что в стеке на момент чтения a^n не меньше, чем $k + 2$ нетерминала. Тогда при подмене w на a^{n+k+1} из стека достанется только k нетерминалов, и слово a^{n+k+1} распознаться не сможет.

Опциональный Else

Рассмотрим GNF-грамматику для языка $\{a^n b^m \mid n \geqslant m\}$. Положим $w = a^{n+k} b^{n+k}$, где n таково, что в стеке на момент чтения a^n не меньше, чем $k + 2$ нетерминала. Тогда при подмене w на a^{n+k+1} из стека достанется только k нетерминалов, и слово a^{n+k+1} распознаться не сможет.

Следствие

Опциональный else не парсится с помощью LL-разбора.

Свойства LL(k)-языков

Утверждение

- LL(k)-языки не замкнуты относительно объединения.
- LL(k)-языки не замкнуты относительно пересечения с регулярным языком (пример: $\{a^n b a^n b\} \cup \{a^n c a^n c\}$ — не LL(k)-язык).
- Если L — LL(k)- нерегулярный язык, то его дополнение — не LL(k) ни для какого k.
- (Более сильное утверждение) Если $L_1 \cup L_2$ — регулярный язык, и L_1 — нерегулярный LL(k)-язык, то L_2 — не LL(k) ни для какого k.