



## Вариант 6, 7, 8

- Найти (в пределе) регулярные язык префиксов и суффиксов, содержащие хотя бы одну итерацию.
- Для каждой пары «накачек» из выбранных регулярных префиксов и суффиксов проверить, что с достаточно высокой вероятностью слова из накачек формируют слово из языка  $\mathcal{L}$ .



## Описание алгоритма

- Определить пределы вывода:  $C$  — допуск на регулярность,  $P$  — число испытаний на эквивалентность,  $P'$  — число испытаний на существование симметричной накачки. Определить  $\Sigma$ .
- Выбрать базу  $\Phi_1 = \{\gamma_1\}$ , где  $\gamma_1 \in \Sigma$  — и вывести язык  $\mathcal{P}(\Phi_1)$  (префиксы в алфавите  $\Phi_1$ ). Для всякого КС-языка его префиксы в унарном алфавите — это регулярный язык, поэтому вывод на этом этапе точный. Если язык  $\mathcal{P}(\Phi_1)$  бесконечен (КА содержит циклы), сохранить его в списке языков префиксов.
- Перейти от  $\Phi_{i-1}$  к  $\Phi_i$  (пробегаая возможные подмножества  $\Sigma$ ). Считаем, что вывод префикса проходит неудачно, если число состояний в автомате для него превышает  $\max_{\gamma_j \in \Sigma} (\Phi_i \setminus \{\gamma_j\}) + C$ . В этом случае считаем  $\Phi_i$  и все содержащие его алфавиты порождающими нерегулярные языки.
- Аналогично порождаем языки суффиксов  $\mathcal{S}(\Phi_i)$ .



## Описание алгоритма

- Строим списки предположительно совместных накачек. Для очередной пары  $\langle \mathcal{P}(\Phi_i), \mathcal{S}(\Phi_j) \rangle$  находим в КА для  $\mathcal{P}(\Phi_i)$  и  $\mathcal{S}(\Phi_j)$  все возможные циклы из состояний в себя (не содержащие итераций по внутренним циклам).
- Пусть  $w_0^p, u_1, w_2^p$  — путь из  $w_0$  до цикла, по циклу, и из цикла в конечное состояние, для языка  $\mathcal{P}(\Phi_i)$ , аналогично —  $w_0^s, v_1, w_2^s$  для  $\mathcal{S}(\Phi_j)$ . Язык  $w_0^p u_1^* w_2^p w_0^s v_1^* w_2^s$  — первичный кандидат на накачку.
- Если с достаточно высокой вероятностью (зависящей от установленного вами параметра  $P'$ ) для выбранного  $k_1$  существует  $k_2$  такое, что  $w_0^p u_1^{k_1} w_2^p w_0^s v_1^{k_2} w_2^s \in \mathcal{L}$ , и для выбранного  $k_2$  существует  $k_1$  такое, что  $w_0^p u_1^{k_1} w_2^p w_0^s v_1^{k_2} w_2^s \in \mathcal{L}$ , тогда объявляем язык  $w_0^p u_1^* w_2^p w_0^s v_1^* w_2^s$  предполагаемой накачкой и возвращаем его.
- В рамках возрастающих алфавитов  $\Phi_i$  возможные варианты накачек (по циклам в автоматах) будут повторяться. Нужно избежать повторов в их проверках.



## Вариант 0, 1, 3, 9

- Дано разбиение  $\langle \omega_1, \omega_2, \omega_3, \omega_4, \omega_5 \rangle$ , предположительно описывающее серию слов  $\omega_1 \omega_2^n \omega_3 \omega_4^n \omega_5$  в языке  $\mathcal{L}$ .
- Найти (в пределе) языки отдельно  $\mathcal{L}(\omega_1)$ ,  $\mathcal{L}(\omega_3)$ ,  $\mathcal{L}(\omega_5)$ .
- Проверить, что пары слов из  $\mathcal{L}(\omega_1)$  и  $\mathcal{L}(\omega_5)$  действительно совместны относительно накачек  $\omega_2$  и  $\omega_4$  в языке  $\mathcal{L}$ .



## Описание алгоритма

- Определить пределы вывода:  $C$  — допуск на регулярность,  $P$  — число испытаний на эквивалентность,  $P'$  — число испытаний на накачку. Определить  $\Sigma$ .
- Проверить, что для достаточно большого числа испытаний  $\omega_1\omega_2^i\omega_3\omega_4^i\omega_5$  действительно принадлежат языку  $\mathcal{L}$ .
- Пусть  $\Phi_1$  — алфавит  $\omega_1$ . Вывести язык префиксов  $p$  для слов вида  $u = \omega_2^i\omega_3\omega_4^i\omega$  такой, что  $pu \in \mathcal{L}$ . Порогом вывода (максимальным размером КА) считать  $|\omega_1| + C$ . Последовательно расширить алфавит выводимых языков префиксов, пока они не превышают порог регулярности (в дальнейшем используя ту же эвристику, что в вариантах 6, 7, 8). Аналогично вывести языки суффиксов  $\omega_5$  и середин  $\omega_3$ . Элементы множества языков префиксов накачки обозначаем  $\mathcal{L}_p$ ; суффиксов —  $\mathcal{L}_s$ .
- Языки «середины накачки» (обобщения  $\omega_3$ ) уже не надо обрабатывать, просто возвращаем их.



## Описание алгоритма

- Язык префиксов  $\mathcal{L}_p$  абсолютно совместен с  $\mathcal{L}_s$ , если  $\forall p \in \mathcal{L}_p, s \in \mathcal{L}_s (p\omega_2^i\omega_3\omega_4^is \in \mathcal{L})$  для достаточно большого числа испытаний степеней  $i$ .
- Если пара  $\langle p_k, s_k \rangle$  нарушает совместность префиксов с суффиксами, сохраняем её в списке контрпримеров. Если число контрпримеров превысило совокупное число состояний в КА для  $\mathcal{L}_p$  и  $\mathcal{L}_s$ , то выводим языки  $\mathcal{L}_{\neg p}$  и  $\mathcal{L}_{\neg s}$  для  $\{p_k\}$  и  $\{s_k\}$ , использующие оракулы  $u \in \mathcal{L}_{\neg p} \Leftrightarrow \exists i, v (u\omega_2^i\omega_3\omega_4^iv \notin \mathcal{L} \ \& \ v \in \mathcal{L}_s)$ ; и  $u \in \mathcal{L}_{\neg s} \Leftrightarrow \exists i, v (v\omega_2^i\omega_3\omega_4^iu \notin \mathcal{L} \ \& \ v \in \mathcal{L}_p)$ . Даже если автоматы языков контрпримеров превысят порог регулярности (размер исходного автомата для  $\mathcal{L}_p$  или  $\mathcal{L}_s$  плюс  $C$ ), всё равно считать вывод языков контрпримеров успешным (в этом случае останавливаемся и уже не ищем новые контрпримеры для КА).
- Построить пересечения  $\mathcal{L}_p \cap \overline{\mathcal{L}_{\neg p}}$  и  $\mathcal{L}_s \cap \overline{\mathcal{L}_{\neg s}}$  (т.е. разности между исходными языками и языками контрпримеров) и вернуть их пару в качестве языков префиксов и суффиксов накачки.



## Вариант 2, 4, 5

- Найти языки разбиений относительно букв алфавита  $\Sigma$  слов языка  $\mathcal{L}$ .
- В рамках этих языков найти неразличимые элементы лексем и последовательные сложные элементы лексем.

В этом варианте будут полезными всевозможные упрощения регулярных выражений, реализованные в предыдущей лабораторной работе. Результаты работы алгоритма будут сильно отличаться для разных базисных наборов лексем, поэтому имеет смысл выбирать такие базисные наборы лексем, которые порождают бесконечные языки относительно разбиений по ним.



## Описание алгоритма

- Определить пределы вывода:  $C$  — допуск на регулярность,  $P$  — число испытаний на эквивалентность. Определить  $\Sigma$ .
- Выбрать базисную лексему  $\gamma_0 \in \Sigma$ . Произвести вывод языка подслов  $\Sigma$ , не содержащих  $\gamma_0$ . Если НКА для языка подслов превысил  $|\Sigma| + C$ , считаем выбор базовой лексемы неудачным и переходим к очередному набору базисных лексем (сначала проверяем все синглтоны, при неудаче — пары базисных лексем, и т.д.). Процесс вывода сходится, поскольку, если выбрать все элементы  $\Sigma$  в качестве базисных лексем, то получится элементарный язык разбиений  $\{\varepsilon\}$ .
- В полученном языке вывести все альтернативы на внешний уровень по дистрибутивности (т.е. привести  $\tau_1(\tau_2 \mid \tau_3)\tau_4$  к виду  $\tau_1\tau_2\tau_4 \mid \tau_1\tau_3\tau_4$ ). Элементы этих альтернатив — потенциальные комбинации лексем. .
- Дальнейшая работа алгоритма производится отдельно с каждым таким элементом (в нашем примере: отдельно с языками  $\tau_1\tau_2\tau_4$  и  $\tau_1\tau_3\tau_4$ ), называемым далее  $\mathcal{L}_{lex}$ .





## Описание алгоритма

- Насыщаем алфавиты лексем. Если символы  $\gamma_i, \gamma_j$  таковы, что замена  $\gamma_i$  на  $\gamma_j$  и наоборот не меняет принадлежность языку  $\mathcal{L}_{\text{lex}}$ , то переходим от  $\mathcal{L}_{\text{lex}}$  к языку, где  $\gamma_i$  и  $\gamma_j$  оба заменены на новую лексему  $\gamma_{\{i,j\}}$ . При этом  $\gamma_{\{i,j\}}$  станет новым символом алфавита  $\Sigma_{\text{lex}}$  языка  $\mathcal{L}_{\text{lex}}$ , а  $\gamma_i$  и  $\gamma_j$  из него удалятся.
- Когда алфавиты лексем будут максимально насыщены, производим отщепление префиксов  $\mathcal{L}_{\text{lex}}$ . А именно, если  $\alpha_i \in \Sigma_{\text{lex}}$  ( $i \leq k$ ), и для всех  $\alpha_i$  частичная производная языка  $\mathcal{L}_{\text{lex}}$  по  $\alpha_i$  включает  $\mathcal{L}_{\text{lex}}$ , тогда строим отображение  $\mathcal{L}_{\text{lex}} = (\alpha_1 \mid \dots \mid \alpha_k)^* \mathcal{L}'_{\text{lex}}$ , где  $\mathcal{L}'_{\text{lex}} = \mathcal{L}_{\text{lex}} \cup \overline{(\alpha_1 \mid \dots \mid \alpha_k)^*}$  (т.е. множество слов  $\mathcal{L}_{\text{lex}}$ , которые не начинаются на  $\alpha_i$ ).
- После отщепления префиксов выявляем сложные лексемы: если частичная производная по слову  $w$  языка  $\mathcal{L}'_{\text{lex}}$  единственна (и не включает  $\mathcal{L}'_{\text{lex}}$  и с пустой регуляркой), а по некоторому слову  $w\gamma$ , где  $\gamma \in \Sigma_{\text{lex}}$ , уже не единственна либо включает себя, тогда считаем  $w$  сложной лексемой (не исключено, что состоящей из одного символа). Переходим от  $\mathcal{L}'_{\text{lex}}$  к производным по всем возможным сложным лексемам и повторяем для них отщепление префиксов и выявление сложных лексем.
- Если не нашлось сложных лексем, возвращаем рассматриваемый язык как единую лексическую единицу.



## Варианты алгоритмов вывода

- (Чётная последняя цифра зачётки) алгоритм  $L^*$ .
- (Нечётная последняя цифра зачётки) алгоритм  $NL^*$ .

Если вы в числе приоритетных (оба сдали ЛР до 16 октября), то можете выбрать в этой позиции любой вариант.



## Оракул эквивалентности

Поскольку сам по себе язык, представляемый оракулом, является контекстно-свободным, то оракул на эквивалентность вероятностный. А именно, он определяется числом  $P$  успешных проверок на случайно порождённых строках вашего ДКА-гипотезы, или на строках, не входящих в язык ДКА (порождать их можно опять же как случайные пути, но с добавлением ловушки, и так, чтобы состояние, в котором автомат оказался после чтения последнего символа, не было финальным). Если у вас вариант с НКА, тогда придётся детерминизировать. Число  $P$  вы определяете самостоятельно, как параметр. Если на  $P$  испытаниях их принадлежность языку гипотезы и целевому языку оказалась одинаковой, то считается, что испытание на эквивалентность успешно пройдено в рамках заданного  $P$ . Если различной — автоматически возвращается контрпример.



## Оракул языка префиксов/подслов

Во всех вариантах требуется оракул не исходного языка, а некоторого подмножества его подслов.

- Самое простое решение — захардкодить, то есть по исходному оракулу построить оракулы для префиксов, суффиксов и инфиксов. Оно допустимо, но следует учесть, что хардкодить можно только собственно такие оракулы, а не пересечения их с регулярными языками, которые представляют собой языки в ограниченных алфавитах, или структуры накачек.
- Если у вас оракул в виде разбора по КС-грамматике, то можно построить грамматики префиксов, инфиксов и суффиксов автоматически. Более того, можно построить вспомогательный модуль, реализующий пересечение этих грамматик с регулярным языком, и тогда в основной части программы запросы удастся проводить в точности к тому языку, который требуется (с поправкой на кванторные условия, о них ниже).
- Информацию, как представлены оракулы (захардкожены или читаются из грамматик), нужно сообщить в readme.



## Оракулы с кванторными условиями

- Если требуется построить оракул для языка  $\{u \mid \exists w(u\omega_2^i\omega_3\omega_4^i w \in \mathcal{L})\}$  (вариант 0, 1, 3, 9) (или аналогичных языков), для его имитации достаточно запускать оракул несколько раз: для префиксов  $\mathcal{L}$  (на собственно  $u$ ) и для проверки, что  $u\omega_2^i\omega_3\omega_4^i$  является префиксом  $\mathcal{L}$  (на определяемом вами числе испытаний для  $i$ ).
- Если требуется построить оракул для языка  $\{u \mid \exists i, v(v \in \mathcal{L}_s \ \& \ u\omega_2^i\omega_3\omega_4^i v \notin \mathcal{L})\}$ , то в качестве одной из возможностей это сделать можно вывести язык  $\{v \mid \exists u(u\omega_2^i\omega_3\omega_4^i v \notin \mathcal{L})\}$  (для ограниченных вами значений  $i$ ) — пользуясь инверсией исходного оракула для  $\mathcal{L}$ , после чего пересечь его с  $\mathcal{L}_s$  и взять из результата тестовые строки  $c_k$  для суффиксов (соответствующие путям из начального состояния в конечное, таким что тестовая строка  $c_k$  соответствует пути, проходящему через состояние  $q_k$ , их число будет соответствовать числу состояний в КА). Оракул для итогового языка: тест, что для какого-нибудь из  $c_k$   $u\omega_2^i\omega_3\omega_4^i c_k \notin \mathcal{L}$ .



## Минимально адекватный учитель (MAT)

MAT отвечает на два типа запросов к оракулу.

- Membership (принадлежность):  $\omega \mapsto \omega \in \mathcal{L}$ .  
Возвращает 1, если  $\omega$  принадлежит  $\mathcal{L}$ , и 0 иначе.
- Equivalence (эквивалентность):  $\mathcal{A} \equiv \mathcal{L}$  (т.е. принимает на вход описание регулярного языка, например, конечным автоматом). Возвращает либо сообщение о том, что эквивалентность выполнена, либо контрпример: слово  $\omega$  такое, что либо  $\omega \in \mathcal{L}(\mathcal{A})$ , но  $\omega \notin \mathcal{L}$ , либо  $\omega \in \mathcal{L}$ , но  $\omega \notin \mathcal{L}(\mathcal{A})$ .



## Расширенная таблица наблюдений

Алгоритмы  $L^*$  и  $NL^*$  строят описание КА при условии обращения к МАТ посредством постепенных приближений таблицы классов эквивалентности. На каждом этапе вычислений таблица должна удовлетворять следующим свойствам:

- полнота — при заглядывании «на шаг вперёд» не должно получаться строк, демонстрирующих иное поведение относительно уже существующих.
- непротиворечивость — если два префикса демонстрируют согласованное поведение в таблице, то при заглядывании «на шаг вперёд» они также должны вести себя согласованно.

Чтобы осуществить требуемое заглядывание, таблица строится из двух частей. Основная состоит из множества строк  $\mathcal{S}$  и столбцов  $\mathcal{E}$ , содержимое таблицы — отметки, принадлежат ли  $uv$  языку  $\mathcal{L}$ , где  $u \in \mathcal{S}$ ,  $v \in \mathcal{E}$ . Расширенная часть — это строки из  $\mathcal{S}$ , не являющиеся префиксами других строк из  $\mathcal{S}$ , с приписанными к ним элементами алфавита  $\Sigma$ , и всё те же столбцы  $\mathcal{E}$ .



## Общая канва алгоритмов $L^*$ и $NL^*$

- 1 Если таблица  $\mathcal{S} \times \mathcal{E}$  неполна, то существует элемент из  $\mathcal{S} \cdot \Sigma$ , который порождает новую строку в таблице.  
Добавляем его в  $\mathcal{S}$  и обновляем расширенную таблицу.
- 2 Если  $\mathcal{S} \times \mathcal{E}$  противоречива, то существует суффикс  $v$  из  $\mathcal{E}$ , показывающий различное поведение строк  $u_1, u_2$  при приписывании к ним суффикса  $\gamma v$  ( $\gamma \in \Sigma$ ).  
Добавляем суффикс  $\gamma v$  в  $\mathcal{E}$  и достраиваем таблицу.
- 3 Когда таблица полна и непротиворечива, строим по ней описание регулярного языка и делаем запрос к МАТ на эквивалентность. Если МАТ вернул контрпример — добавляем его и все его префиксы в множество  $\mathcal{S}$  и продолжаем процесс. Вариант: добавить контрпример и все его суффиксы в  $\mathcal{E}$ .
- 4 Результат — описание регулярного языка, которое признаётся оракулом как корректное.





## Алгоритм $L^*$

- Условие полноты — отсутствие в  $\mathcal{S}.\Sigma \times \mathcal{E}$  строк, которые отличаются от строк в  $\mathcal{S} \times \mathcal{E}$ .
- Условие непротиворечивости — отсутствие в  $\mathcal{S}.\Sigma \times \mathcal{E}$  таких позиций  $i, j, k$ , что  $u_i v_k \neq u_j v_k$ , притом что  $u_i = u'_i \gamma$ ,  $u_j = u'_j \gamma$ , и строки в таблице  $\mathcal{S} \times \mathcal{E}$  для  $u'_i$  и  $u'_j$  совпадают.
- ДКА по таблице строится следующим образом.
  - Состояния ДКА — кратчайшие слова из  $\mathcal{S}$ , порождающие разные строки в  $\mathcal{S} \times \mathcal{E}$ .
  - Начальное состояние соответствует префиксу  $\varepsilon$ .
  - Конечные состояния — те, которые содержат 1 в столбце, помеченном  $\varepsilon$ .
  - Если  $u\gamma \equiv u'$  (т.е.  $u\gamma$  и  $u'$  соответствуют одинаковым строчкам в таблице), то  $\langle u, \gamma \rangle \rightarrow u'$  добавляется в ДКА как переход.



## Алгоритм $NL^*$

Алгоритм строит минимальный остаточный НКА (RFSA).

Скажем, что строка  $r$  накрывается  $\bigcup_k r_k$ , если  $\forall i (r[i] = \bigvee_k r_k[i])$  (т.е. она является поэлементной дизъюнкцией строк  $r_k$ ).

Строка  $r_1$  поглощает  $r_2$  ( $r_2 \sqsubseteq r_1$ ), если  $\forall i (r_1[i] \geq r_2[i])$ .

- Условие полноты — отсутствие в  $\mathcal{S} \cdot \Sigma \times \mathcal{E}$  строк, которые не накрываются строками в  $\mathcal{S} \times \mathcal{E}$ .
- Условие непротиворечивости — отсутствие в  $\mathcal{S} \times \mathcal{E}$  таких позиций  $i, j, k$ , что  $u_i \sqsubseteq u_j$ , но для некоторого  $\gamma \in \Sigma$   $u_i \gamma \not\sqsubseteq u_j \gamma$  в расширенной таблице.
- НКА по таблице строится следующим образом.
  - Состояния НКА — кратчайшие слова из  $\mathcal{S}$ , базисные (т.е. не накрываемые набором других) в  $\mathcal{S} \times \mathcal{E}$ .
  - Начальные состояния включают строки, поглощаемые строкой  $\varepsilon$  (чтобы перейти к классическому НКА, придётся стянуть их в одно).
  - Конечные состояния — те, которые содержат 1 в столбце, помеченном  $\varepsilon$ .
  - Если  $v \sqsubseteq u\gamma$ , то  $\langle u, \gamma \rangle \rightarrow v$  добавляется в НКА как переход.