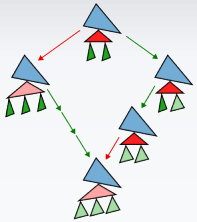
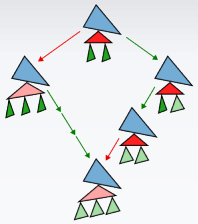


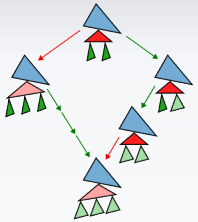
Вариант 6, 7, 8

- 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 $\mathscr{L}$.



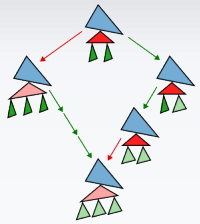
Описание алгоритма

- Определить пределы вывода: C — допуск на регулярность, P — число испытаний на эквивалентность, P' — число испытаний на существование симметричной накачки. Определить Σ.
- Выбрать базу $\Phi_1 = \{\gamma_1\}$, где $\gamma_1 \in \Sigma$ — и вывести язык $\mathcal{P}(\Phi_1)$ (префиксы в алфавите Φ_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 Φ_{i-1} к Φ_i (пробегая возможные подмножества Σ). 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 Φ_i и все содержащие его алфавиты порождающими нерегулярные языки.
- Аналогично порождаем языки суффиксов $\mathcal{S}(\Phi_i)$.



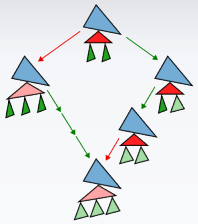
Описание алгоритма

- 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 $w_0^p u_1^{k_1} w_2^p w_0^s v_1^{k_2} w_2^s \in \mathscr{L}$, и для выбранного k_2 существует k_1 такое, что $w_0^p u_1^{k_1} w_2^p w_0^s v_1^{k_2} w_2^s \in \mathscr{L}$, 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 Φ_i возможные варианты накачек (по циклам в автоматах) будут повторяться. Нужно избежать повторов в их проверках.



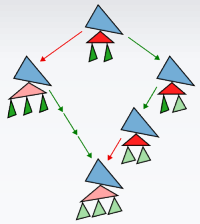
Вариант 0, 1, 3, 9

- 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 $\mathscr{L}$.
- Найти (в пределе) языки отдельно $\mathscr{L}(\omega_1)$, $\mathscr{L}(\omega_3)$, $\mathscr{L}(\omega_5)$.
- Проверить, что пары слов из $\mathscr{L}(\omega_1)$ и $\mathscr{L}(\omega_5)$ действительно совместны относительно накачек ω_2 и ω_4 в языке $\mathscr{L}$.



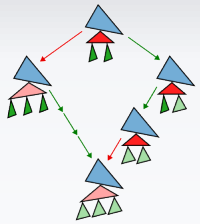
Описание алгоритма

- Определить пределы вывода: C — допуск на регулярность, P — число испытаний на эквивалентность, P' — число испытаний на накачку. Определить Σ.
- Проверить, что для достаточно большого числа испытаний $\omega_1\omega_2^i\omega_3\omega_4^i\omega_5$ действительно принадлежат языку $\mathscr{L}$.
- Пусть Φ_1 — алфавит ω_1. Вывести язык префиксов p для слов вида $u = \omega_2^i\omega_3\omega_4^i w$ такой, что $pu \in \mathscr{L}$. 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 ω_5 и середин ω_3. Элементы множества языков префиксов накачки обозначаем $\mathscr{L}_p$; суффиксов — $\mathscr{L}_s$.
- Языки «середины накачки» (обобщения ω_3) уже не надо обрабатывать, просто возвращаем их.



Описание алгоритма

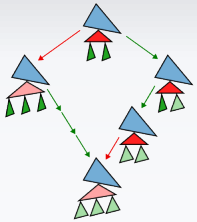
- Язык префиксов $\mathscr{L}_p$ абсолютно совместен с $\mathscr{L}_s$, если $\forall p \in \mathscr{L}_p, s \in \mathscr{L}_s (p\omega_2^i\omega_3\omega_4^i s \in \mathscr{L})$ для достаточно большого числа испытаний степеней i.
- Если пара $\langle p_k, s_k \rangle$ нарушает совместность префиксов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



Вариант 2, 4, 5

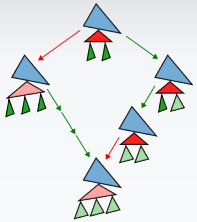
- Найти языки разбиений относительно букв алфавита Σ слов языка $\mathscr{L}$.
- В рамках этих языков найти неразличимые элементы лексем и последовательные сложные элементы лексем.

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.



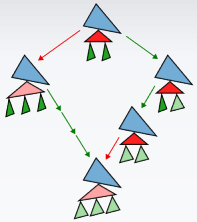
Описание алгоритма

- Определить пределы вывода: C — допуск на регулярность, P — число испытаний на эквивалентность. Определить Σ.
- Выбрать базисную лексему $\gamma_0 \in \Sigma$. Произвести вывод языка подслов Σ, не содержащих γ_0. Если НКА для языка подслов превысил $|\Sigma| + C$, считаем выбор базовой лексемы неудачным и переходим к очередному набору базисных лексем (сначала проверяем все синглетоны, при неудаче — пары базисных лексем, и т.д.). Процесс вывода сходится, поскольку, если выбрать все элементы Σ 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 $\mathscr{L}_{\mathtt{lex}}$.



Описание алгоритма

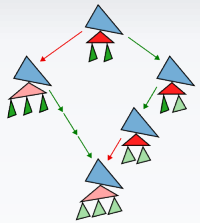
- Насыщаем алфавиты лексем. Если символы γ_i, γ_j таковы, что замена γ_i на γ_j и наоборот не меняет принадлежность языку $\mathscr{L}_{lex}$, то переходим от $\mathscr{L}_{lex}$ к языку, где γ_i и γ_j оба заменены на новую лексему $\gamma_{\{i,j\}}$. При этом $\gamma_{\{i,j\}}$ станет новым символом алфавита Σ_{lex} языка $\mathscr{L}_{lex}$, а γ_i и γ_j из него удалятся.
- Когда алфавиты лексем будут максимально насыщены, производим отщепление префиксов $\mathscr{L}_{lex}$. А именно, если $\alpha_i \in \Sigma_{lex}$ $(i \leqslant k)$, и для всех α_i частичная производная языка $\mathscr{L}_{lex}$ по α_i включает $\mathscr{L}_{lex}$, тогда строим отображение $\mathscr{L}_{lex} = \underline{(\alpha_1 \mid \ldots \mid \alpha_k)^* \mathscr{L}'_{lex}}$, где $\mathscr{L}'_{lex} = \mathscr{L}_{lex} \cup (\alpha_1 \mid \ldots \mid \alpha_k).^*$ (т.е. множество слов $\mathscr{L}_{lex}$, которые не начинаются на α_i).
- После отщепления префиксов выявляем сложные лексемы: если частичная производная по слову w языка $\mathscr{L}'_{lex}$ единственна (и не включает $\mathscr{L}'_{lex}$ и с пустой регуляркой), а по некоторому слову $w\gamma$, где $\gamma \in \Sigma_{lex}$, уже не единственна либо включает себя, тогда считаем w сложной лексемой (не исключено, что состоящей из одного символа). Переходим от $\mathscr{L}'_{lex}$ 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`*.

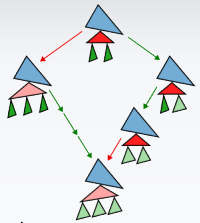
Если вы в числе приоритетных (оба сдали ЛР до 16 октября), то можете выбрать в этой позиции любой вариант.



Минимально адекватный учитель (MAT)

MAT отвечает на два типа запросов к оракулу.

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

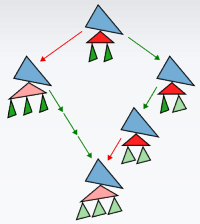


Расширенная таблица наблюдений

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

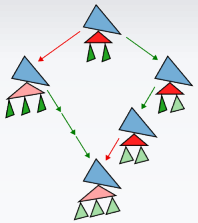
- полнота — при заглядывании «на шаг вперёд» не должно получаться строк, демонстрирующих иное поведение относительно уже существующих.
- непротиворечивость — если два префикса демонстрируют согласованное поведение в таблице, то при заглядывании «на шаг вперёд» они также должны вести себя согласованно.

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 $\mathfrak{u}\mathfrak{v}$ языку $\mathscr{L}$, где $\mathfrak{u} \in \mathcal{S}$, $\mathfrak{v} \in \mathcal{E}$. Расширенная часть — это строки из $\mathcal{S}$, не являющиеся префиксами других строк из $\mathcal{S}$, с приписанными к ними элементами алфавита Σ, и всё те же столбцы $\mathcal{E}$.



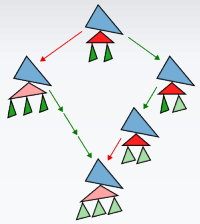
Общая канва алгоритмов L^* и NL^*

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



Алгоритм L^*

- Условие полноты — отсутствие в $S.\Sigma \times E$ строк, которые отличаются от строк в $S \times E$.
- Условие непротиворечивости — отсутствие в $S.\Sigma \times E$ 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 $S \times E$ для u_i' и u_j' совпадают.
- ДКА по таблице строится следующим образом.
  - Состояния ДКА — кратчайшие слова из S, порождающие разные строки в $S \times E$.
  - Начальное состояние соответствует префиксу ε.
  - Конечные состояния — те, которые содержат 1 в столбце, помеченном ε.
  - Если $u\gamma \equiv u'$ (т.е. $u\gamma$ и u' соответствуют одинаковым строчкам в таблице), то $\langle u, \gamma \rangle \to u'$ добавляется в ДКА как переход.



Алгоритм $\mathtt{NL}^*$

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 $\forall i(r_1[i] \geqslant r_2[i])$.

- Условие полноты — отсутствие в $\mathcal{S}.\Sigma \times \mathcal{E}$ 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 накрывающиеся набором других) в $\mathcal{S} \times \mathcal{E}$.
  - Начальные состояния включают строки, поглощаемые строкой ε (чтобы перейти к классическому НКА, придётся стянуть их в одно).
  - Конечные состояния — те, которые содержат 1 в столбце, помеченном ε.
  - Если $v \sqsubseteq u\gamma$, то $\langle u, \gamma \rangle \to v$ добавляется в НКА как переход.