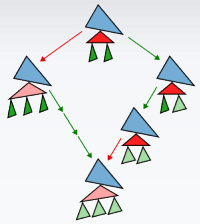
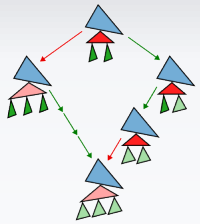


Краткое резюме задачи

Каждая задача состоит из двух частей: конвертация и тестирование.

- (0,1,3,9) Приведение shuffle-регулярных выражений к академическим.
- (2,4,5) Оптимизация неэффективностей в академических регулярных выражениях.
- (6,7,8) Приведение lookahead-регулярных выражений к академическим.

Тестирование основано на принципе фаззинга: порождения случайных выражений и строк, которые должны распознаваться результатом преобразования точно так же, как и исходным выражением. В п.2 фазз-тестирование дополнительно использует принцип «malicious pump» — злонамеренной накачки.



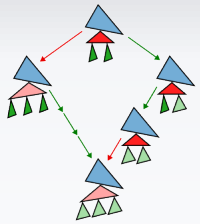
Fuzz-модуль

В fuzz-модуле должны быть следующие составляющие:

- Генератор регулярок
- Генератор строк
- Сравнительный парсинг строки по регуляркам

Генератор строк использует автоматное представление выражения — поэтому даже вариантам 2,4,5 без построения автомата по регулярке не обойтись. У остальных эта опция и так является частью задачи.

В последнем пункте предпочтительно пользоваться библиотеками регулярных выражений. У варианта 0,1,3,9 для регулярок с shuffle-операцией такой возможности не будет, поэтому придётся написать парсер: по автомату Брзозовски будет просто, по автомату Антимирова — чуть хуже, потому что придётся отслеживать пути недетерминированно.

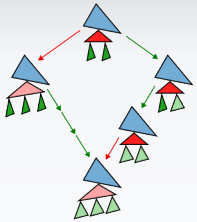


Рандомизация регулярок

Следующие параметры должны быть вынесены в макросы или ключи:

1. размер алфавита (можно предполагать, что он не больше 5 — т.к. автоматы для больших алфавитов превратятся в «ёжики» без факторизации по классам букв, а в **этом году** задача не про неё);
2. звёздная высота (число вложенных квантификаторов итераций);
3. (для вариантов 6,7,8) число lookahead-ов;
4. максимальное число букв в регулярке.

Предполагаем для простоты, что выражения $(a|)$ (т.е. такие, у которых аргументом альтернативы является пустое выражение) невозможны, как невозможны и выражения вида $()^*$ (итерация над пустым выражением). Дальше на каждом шаге генерации можно рандомно выбирать очередной применяемый оператор (если в запасе есть хотя бы 2 буквы) либо константу, либо итерацию, а затем рекурсивно делать генерацию по его аргументам, учитывая, что запас свободных букв при генерации очередной бинарной операции заведомо уменьшается хотя бы на 1.



Генератор строк

Случайные строки с вероятностью почти 1 интереса для фазз-тестирования не представляют. Обычно такие генераторы используют генетические алгоритмы, порождающие «правдоподобные» вводы, но мы ограничимся комбинаторным подходом.

- Строится матрица достижимости в автомате, построенном на базе регулярного выражения.
- В рамках отношения достижимости выбирается случайная последовательность состояний $\{q_i\}$ таких, что q_{i+1} достижимо из q_i, q_0 — стартовое, последнее состояние — финальное.
- Посредством, например, BFS строятся слова на путях из q_i в q_{i+1}. Если автомат недетерминированный, рекомендуется использовать недетерминированные структуры данных.
- К полученным отрезкам применяются случайные мутации: перестановка букв, перестановка фрагментов, повторение букв, повторение фрагментов, удаление букв, удаление фрагментов (либо не применяется никаких).

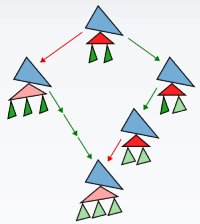


Схема преобразования Shuffle-выражений

- Построить автомат Брзозовски или Антимирова для shuffle-регулярного выражения.
- Преобразовать его в классическое регулярное выражение.

Правила задания shuffle-операции и производной для неё следующие:

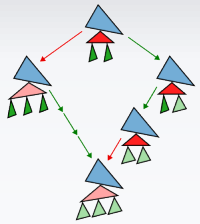
$$r \# \varepsilon = r$$
$$r \# \varnothing = \varnothing$$
$$ar_1 \# br_2 = a(r_1 \# br_2) \mid b(ar_1 \# r_2)$$

Производная Брзозовски:

$$a^{-1}(r_1 \# r_2) = a^{-1}(r_1) \# r_2 \mid r_1 \# a^{-1}(r_2)$$

Производная Антимирова:

$$\alpha_a(r_1 \# r_2) = \{\alpha_a(r_1) \# r_2\} \cup \{r_1 \# \alpha_a(r_2)\}$$



ACI-нормализация

В случае построения автомата Брзозовски регулярные выражения требуется упрощать по ACI (ассоциативности, коммутативности, идемпотентности) альтернативы.

- Все ассоциативности делаем либо левосторонними (если используется бинарное дерево), либо плоскими (если используется произвольное дерево либо Рефал-стиль).
- Сортируем аргументы альтернативы по лексикографическому возрастанию.
- Удаляем одинаковые аргументы альтернатив.

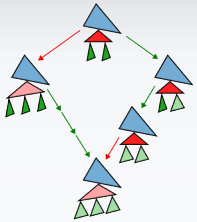
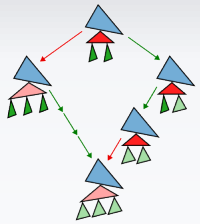


Схема нормализации regex

- Сделать SSNF-преобразование.
- Вынести общие множители за скобки слева и справа (рекурсивно). То есть до исчерпания применить правила DSTR (левое и правое) алгебры Клини к выражениям, нормализованным по ACI (см. предыдущий вариант).
- В фазз-тестировании оценить сравнительное быстродействие результатов.



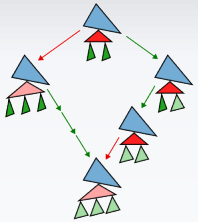
Сильная звёздная нормальная форма

Алгоритм приведения в SSNF определяется рекурсивно:

- $\beta = \varepsilon \Rightarrow \mathrm{ssnf}(\beta) = \varepsilon$;
- $\beta = a \Rightarrow \mathrm{ssnf}(\beta) = a$;
- $\beta = A \mid B \Rightarrow \mathrm{ssnf}(\beta) = \mathrm{ssnf}(A) \mid \mathrm{ssnf}(B)$;
- $\beta = AB \Rightarrow \mathrm{ssnf}(\beta) = \mathrm{ssnf}(A)\,\mathrm{ssnf}(B)$;
- $\beta = A^* \Rightarrow \mathrm{ssnf}(\beta) = \mathrm{ss}(A)^*$.

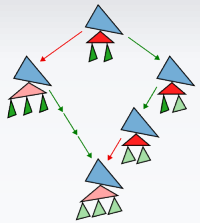
Для преобразования подзвездного выражения используется следующий алгоритм (запись вида β_ε означает, что регулярное выражение β порождает пустое слово ε):

- $\beta = \varepsilon \Rightarrow \mathrm{ss}(\beta) = \varnothing$;
- $\beta = a \Rightarrow \mathrm{ss}(\beta) = a$;
- $\beta = A \mid B \Rightarrow \mathrm{ss}(\beta) = \mathrm{ss}(A) \mid \mathrm{ss}(B)$;
- $\beta = A_\varepsilon B_\varepsilon \Rightarrow \mathrm{ss}(\beta) = \mathrm{ss}(A_\varepsilon) \mid \mathrm{ss}(B_\varepsilon)$;
- $\beta = AB \Rightarrow \mathrm{ss}(\beta) = \mathrm{ssnf}(A)\,\mathrm{ssnf}(B)$;
- $\beta = A^* \Rightarrow \mathrm{ss}(\beta) = \mathrm{ss}(A)$.



Злонамеренная накачка

- Построить автомат Глушкова для регулярного выражения (рекомендуется, чтобы не мучиться потом с ε-переходами).
- Реализовать стандартный фазз-тест на эквивалентность языков.
- Породить фрагменты путей: q_0, q_1, q_1, q_2, q_2 (q_i здесь не обязаны быть различными). То есть при выборе состояний q_i брать только те, которые достижимы из себя. Соответствующие фрагменты — это A_{01}, A_{11}, A_{12}, A_{22}. Если состояний, которые достижимы из себя, нет, то объявить, что регулярка нециклическая.
- Построить «злонамеренную накачку»: слово $\Theta = A_{01}A_{11}^{100}A_{12}A_{22}\gamma$, где γ — буква, которой нет в алфавите регулярки.
- Запустить стандартный метод сопоставления по регулярному выражению на старой и новой регулярке, используя строку Θ, и сравнить время его исполнения.



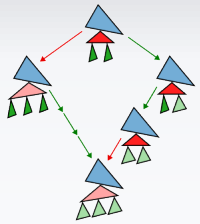
Преобразование lookahead-выражений

- Блочная сборка автомата-распознавателя из подавтоматов с использованием алгоритма пересечения автоматов.
- Преобразование автомата обратно в регулярное выражение.

Для простоты вводятся следующие ограничения синтаксиса:

- lookahead нельзя использовать внутри альтернативы, вслед за которой есть конкатенация (то есть если в альтернативе есть lookahead, то за этой альтернативой может идти только конец строки).
- нельзя использовать lookahead под итерацией.

То есть выражение
$\hat{}\texttt{ab}((?=.*(\texttt{aa}|\texttt{b})\$)(\texttt{a}|\texttt{ab})^* \mid (?=.*(\texttt{ba}|\texttt{aa})\$)(\texttt{b}|\texttt{ba})^*)\$$
допустимо в качестве входного выражения, а
$\hat{}((?=.*(\texttt{aa}|\texttt{b})\$)(\texttt{a}|\texttt{ab})^* \mid (?=.*(\texttt{ba}|\texttt{aa})\$)(\texttt{b}|\texttt{ba})^*)\texttt{ab}\$$ нет, из-за суффикса `ab` после альтернативы, содержащей lookahead.



Блочная сборка НКА по выражению

- Фрагменты выражения, не содержащие lookahead (в том числе выражения внутри lookahead-фрагментов), преобразуются к автоматам Глушкова.
- Реализуются операции конкатенации, объединения и пересечения НКА без ε-переходов и без переходов в начальное состояние (в случае конкатенации и объединения можно пользоваться матричной схемой сборки автомата Глушкова).
- Рассмотрим выражение $r = r_1(? = r_2)r_3$, где r_1 не содержит lookahead-операций. Пусть префикс r_1 преобразован в $\mathscr{A}_1$, lookahead выражение r_2 — в $\mathscr{A}_2$, а остаток r_3 — в автомат $\mathscr{A}_3$. Тогда НКА для выражения r — это автомат $\mathscr{A}_1(\mathscr{A}_2 \cap \mathscr{A}_3)$.