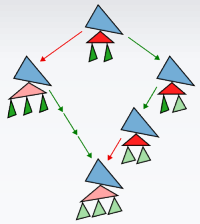
Дополнительное задание на +3

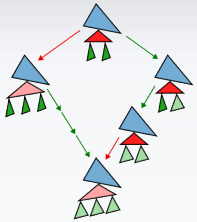
Задание полностью обесценится после 9 октября. 3 балла не входят в стоимость основной л.р. За списанные регулярки в пункте 2 буду нещадно собирать компромат.

1. Выяснить, какой алгоритм парсинга используется в какой-нибудь regex-библиотеке языка из верхней правой четверти графика RedMonk. Описать его на $\frac{1}{2}$–$\frac{2}{3}$ страницы.
2. Протестировать выбранную библиотеку на следующих регулярных выражениях: `((((a)*a)*)*...a)*` (k раз итерируется операция приписывания буквы `a` к предыдущему выражению и применение к результату итерации Клини), `(a?)(a?)...(a?)aa..a` (k раз `a?`, и затем k раз просто `a`), `(.)*b(.)[n]` (n произвольных букв после буквы `b`); а также двух сериях регулярных выражений на ваш выбор. Построить график зависимости скорости распознавания слова от параметра k (значения параметров лучше брать с шагом, $\geqslant 3$, или хотя бы $\geqslant 2$).



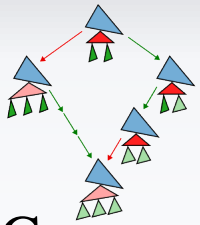
Коллективная часть (подробности ТА)

1. Построение производных и частичных производных, определение длины накачки;
2. Построение НКА по классическим алгоритмам (автоматы Глушкова, Томпсона, Антимирова, follow-автомат);
3. Базовые алгоритмы над НКА и ДКА (детерминизация, удаление эпсилон-правил, удаление/добавление ловушки, минимизация, обращение, отрицание, пересечение, ...);
4. Проверка автоматов на trace equivalence, бисимилярность, эквивалентность их языков (последнее — для контроля). Проверка на синхронизируемость;
5. Реализация парсинга слова по НКА (алгоритм Пайка + стандартный алгоритм с откатами);
6. Интеграция с лабораторными работами прошлого года (преобразование автомата в regex) и с лабораторными работами этого года (трансформационный моноид, нормализация по списку правил).



Индивидуальные варианты

- Вариант $= 0 \mod 3$: нормализация regex по пользовательским правилам переписывания.
- Вариант $= 1 \mod 3$: построение регулярных подвыражений для regex.
- Вариант $= 2 \mod 3$: построение трансформационного моноида.



Нормализация regex: синтаксис

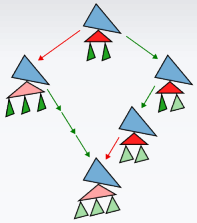
Синтаксис правил переписывания (в нелинейной форме):

<rule> ::= <regex> = <regex>
<regex> ::= (<regex><binary><regex>)
| <symbol><unary> | (<regex>)<unary> | ε
<binary> ::= | | ε
<unary> ::= * | ε

Ассоциативность не предполагается. "Унарный" ε — это отсутствие операции (введено для сокращения записи правил синтаксиса). Пустая бинарная операция — конкатенация.

Вводимое регулярное выражение имеет синтаксис <regex>. Если в правилах переписывания встречаются строчные латинские буквы, которых нет во входном регулярном выражении, их следует понимать как переменные (и только их).

Общая форма входа: <regex> (пустая строка) <rule>*. Либо можно читать выражение и список правил из разных входных файлов (вариант: что-то из потока входа, что-то из файла).

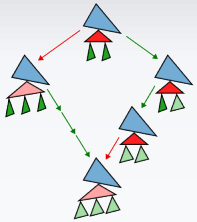


Нормализация: алгоритм

1. Если никакой подтерм выражения не унифицируется с левой частью никакого правила — нормализация завершена.
2. Иначе выбираем самое первое правило в списке, которое унифицируется с некоторым подтермом. Выбираем самый внешний подтерм регулярного выражения, удовлетворяющий условиям унификации, после чего осуществляем перестройку дерева, подставив вместо него правую часть правила, в которой все переменные заменены на их значения, полученные унификацией.

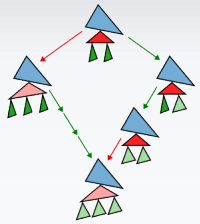
Результатом работы программы является нормализованное регулярное выражение либо зацикливание.

В данном алгоритме используется ограниченная унификация: одно из унифицируемых выражений не содержит переменных.



Нормализация: детали

- Авторы самых быстрых алгоритмов из класса работающих корректно и сданных в срок (причём минимум за 12 часов до конца нештрафного срока) получит от +1 до +3 баллов бонуса.
- Можно подключить проверку правил переписывания на завершаемость с помощью условия Кнута–Бендикса (для этого необходимо доопределить лексикографический порядок на словаре регулярного выражения) и выдавать пользователю соответствующее предупреждение. +1 балл, для сдавших в срок, и +3 балла, если на вашем языке Кнута–Бендикса не сдавали.
- Если вы пишете на языке, на котором нет реализаций Мартелли-Монтанари в первой лабораторной, то получаете ещё +1 балл.



Построение регулярных подвыражений

На вход алгоритму подаётся единственное регулярное выражение R.

Синтаксис входных данных несколько иной, чем в нулевом варианте (есть ассоциативные операторы, допустима позитивная итерация):

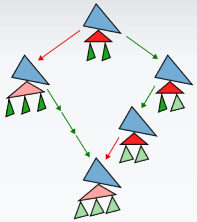
<regex> ::= <regex><binary><regex> | (<regex>)
| <regex><unary> | <symbol> | ε

<binary> ::= | | ε

<unary> ::= * | +

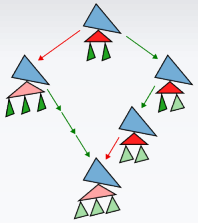
Приоритет операций: * > конкатенация > альтернатива.

Выход алгоритма: список регулярных выражений T_i таких, что $\exists u, v (u\, L(T_i)\, v = L(R))$. При этом среди регулярных выражений T_i могут встречаться выражения, описывающие эквивалентные друг другу языки.



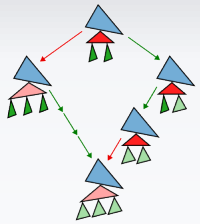
Описание алгоритма

1. Нулевой шаг: построение словаря выражения и задание на нём лексикографического порядка, необходимого для ACI-упрощений.
2. Первый шаг: построение автомата Брзозовски для R. Состояния автомата определят все суффиксные регулярные выражения S_i для R.
3. Второй шаг: построение автоматов Брзозовски для S_i^R. Их состояния $Q_{i,j}$ определят реверсы искомых T_i.
4. Заключительный шаг — обращение $Q_{i,j}$ и удаление из итогового списка регулярных выражений дубликатов.



Детали

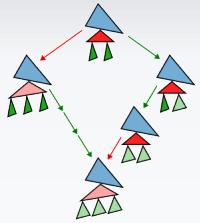
- Чтобы конструкция автомата Брзозовски была конечной, нужно использовать ACI-правила упрощения регулярных выражений, чтобы все состояния всех автоматов были нормализованы. Поскольку | вводится как ассоциативная операция, следует преобразовать входное выражение к неассоциативной (относительно альтернативы) форме.
- В итоговом списке могут быть выражения, описывающие один и тот же язык, и выражения T_i такие, что их язык вкладывается в язык некоторых других T_j (т.е. избыточные). Авторы алгоритмов, реализующих вывод результатов в наиболее лаконичных формах, получат бонусы от +1 до +3 баллов (из работающих корректно и сданных в срок — см. вариант номер 0).



Ещё об интеграции

Упрощение регулярных подвыражений тесно связано с нулевой задачей — переписывание по пользовательским правилам. Если вы используете переписывающий алгоритм вашего коллеги и подберёте удачные упрощающие правила для представления результата, верифицированные с точки зрения завершаемости — +2 балла получат оба (но только один раз, и только сдавшие в срок, см. вариант номер 0).

Если при этом результаты анализа на подвыражения будут встроены в групповой проект, то капитан группы получит +1 балл, дизайнер тоже +1, интегратор +2 балла, и оба участника индивидуальных проектов — по баллу.



Трансформационный моноид

На вход алгоритма подаётся детерминированный конечный автомат (то, что он детерминированный, необходимо проверить, и в противном случае вывести сообщение о некорректности входных данных).

Первая строка автомата — это перечисление начального состояния (первый элемент кортежа) и множества конечных состояний (второй элемент, т.е. в фигурных скобках). Далее вводятся правила перехода.

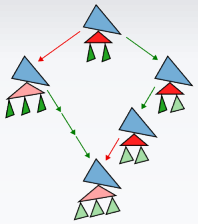
⟨First line⟩ ::= <⟨state⟩,{⟨state⟩(,⟨state⟩)* }>

⟨transition⟩ ::= <⟨state⟩,⟨letter⟩>->⟨state⟩

⟨state⟩ ::= [A–Z][0-9]?

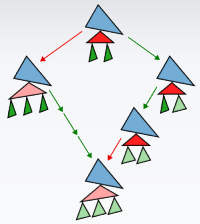
⟨letter⟩ ::= [a–z]

В автомате могут присутствовать состояния-ловушки, и состояния, не достижимые из начального состояния. Такие состояния необходимо удалить.



Т.М.: алгоритм

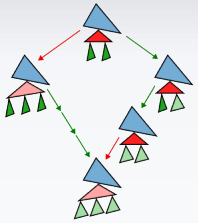
- Определяем лексикографический порядок на входном алфавите автомата.
- Пока хотя бы одно слово внутри итерации вызывает какие-нибудь изменения в моноиде, перебираем все слова длины k, упорядоченные лексикографически, с увеличением k:
  1. если очередное слово w упрощается с помощью хотя бы одного уже построенного правила переписывания, переходим к следующему слову в списке.
  2. иначе строим множество пар $M_w = \{(q_i, q_j)\}$ таких, что $q_i \xrightarrow{w} j$. Если $M_w = M_v$ для какого-то из ранее рассмотренных v, то добавляем правило $w \to v$ в моноид; если M_w уникально, то добавляем w в классы эквивалентности моноида.



Т.М.: анализ результатов

Помимо списка правил моноида и множества его классов эквивалентности $\{w_i\}$ относительно указанных правил, требуется следующее:

- список классов эквивалентности w_i таких, что $w_i \in L(\mathscr{A})$ (т.е. входят в язык автомата).
- для каждого класса w:
  - список классов эквивалентности v_i таких, что $v_i\, w \in L(\mathscr{A})$;
  - список классов эквивалентности v_i таких, что $w\, v_i \in L(\mathscr{A})$;
  - список пар $\langle v_i, v_j \rangle$ таких, что $v_i\, w\, v_j \in L(\mathscr{A})$;
  - состояние q_i, к которому w синхронизирует $\mathscr{A}$, либо сообщение, что w не синхронизирующее слово.



Т.М.: детали

1. Самая быстрая среди корректных и сданных в срок реализация получит от +1 до +3 бонусных балла.
2. Если дополнительно $\mathscr{A}$ будет тестироваться на минимальность и в случае удачи из трансформационного моноида будет извлекаться информация о классах эквивалентности по Майхиллу–Нероуду (с перечислением представителей этих классов и суффиксов, которые их различают, в форме таблицы), это добавит 2 балла (касается сдающих в срок).
3. Интеграция этой лабораторной работы в работы групп (как дополнительной) добавит +2 балла сдающему, +1 балл капитану, +1 балл дизайнеру группы и +2 балла интегратору.