Автомат Глушкова

Линеаризация

Определение

Если регулярное выражение $r \in \mathcal{RE}$ содержит n вхождений букв алфавита Σ, тогда линеаризованное регулярное выражение $\mathtt{Linearize}(r)$ получается из r приписыванием i-ой по счёту букве, входящей в r, индекса i.

Пример

Рассмотрим регулярное выражение:

$$(\mathtt{ba} \mid \mathtt{b})\mathtt{aa}(\mathtt{a} \mid \mathtt{ab})^*$$

Его линеаризованная версия:

$$(\mathtt{b}_1\mathtt{a}_2 \mid \mathtt{b}_3)\mathtt{a}_4\mathtt{a}_5(\mathtt{a}_6 \mid \mathtt{a}_7\mathtt{b}_8)^*$$

Множества First, Last, Follow

Определение

Пусть $r \in \mathcal{RE}$, тогда:

- множество First — это множество букв, с которых может начинаться слово из $\mathscr{L}(r)$ (если $\varepsilon \in \mathscr{L}(r)$, то оно формально добавляется в First);
- множество Last — это множество букв, которыми может заканчиваться слово из $\mathscr{L}(r)$;
- множество Follow(c) — это множество букв, которым может предшествовать c. Т.е. $\{d \in \Sigma \mid \exists w_1, w_2 (w_1 c d w_2 \in \mathscr{L}(r))\}$.

Множество Follow в теории компиляции обычно определяется иначе — это множество символов, которые могут идти за выводом из определённого нетерминального символа. Два этих определения можно унифицировать, если рассматривать каждую букву в r как «обёрнутую»

First, Last, Follow — **пример**

Построим указанные множества для регулярного выражения $\mathtt{r} = (\mathtt{ba} \mid \mathtt{b})\mathtt{aa}(\mathtt{a} \mid \mathtt{ab})^*$.
Начнём с исходного регулярного выражения.

Исходное регулярное выражение

- $\mathsf{First}(\mathtt{r}) = \{\mathtt{b}\}$.
- $\mathsf{Last}(\mathtt{r}) = \{\mathtt{a}, \mathtt{b}\}$.
- $\mathsf{Follow}_{\mathtt{r}}(\mathtt{a}) = \{\mathtt{a}, \mathtt{b}\}$; $\mathsf{Follow}_{\mathtt{r}}(\mathtt{b}) = \{\mathtt{a}\}$.

Хотя данные множества описывают, как устроены слова из $\mathscr{L}(\mathtt{r})$ локально, однако они не исчерпывают всей информации о языке, поскольку разные вхождения букв в регулярное выражения никак не различаются.
Например, по множествам First и Last можно предположить, что $\mathtt{b} \in \mathscr{L}(\mathtt{r})$, хотя это не так.

First, Last, Follow — **пример**

Построим указанные множества для регулярного выражения $\mathtt{r} = (\mathtt{ba} \mid \mathtt{b})\mathtt{aa}(\mathtt{a} \mid \mathtt{ab})^*$.
Вспомним, что $\mathtt{r}_{\mathsf{Lin}} = (\mathtt{b}_1\mathtt{a}_2 \mid \mathtt{b}_3)\mathtt{a}_4\mathtt{a}_5(\mathtt{a}_6 \mid \mathtt{a}_7\mathtt{b}_8)^*$.

Линеаризованное выражение

- $\mathsf{First}(\mathtt{r}_{\mathsf{Lin}}) = \{\mathtt{b}_1, \mathtt{b}_3\}$.
- $\mathsf{Last}(\mathtt{r}_{\mathsf{Lin}}) = \{\mathtt{a}_5, \mathtt{a}_6, \mathtt{b}_8\}$.
- $\mathsf{Follow}_{\mathtt{r}_{\mathsf{Lin}}}(\mathtt{b}_1) = \{\mathtt{a}_2\}$; $\mathsf{Follow}_{\mathtt{r}_{\mathsf{Lin}}}(\mathtt{a}_2) = \{\mathtt{a}_4\}$;
  $\mathsf{Follow}_{\mathtt{r}_{\mathsf{Lin}}}(\mathtt{b}_3) = \{\mathtt{a}_4\}$; $\mathsf{Follow}_{\mathtt{r}_{\mathsf{Lin}}}(\mathtt{a}_4) = \{\mathtt{a}_5\}$;
  $\mathsf{Follow}_{\mathtt{r}_{\mathsf{Lin}}}(\mathtt{a}_5) = \{\mathtt{a}_6, \mathtt{a}_7\}$; $\mathsf{Follow}_{\mathtt{r}_{\mathsf{Lin}}}(\mathtt{a}_6) = \{\mathtt{a}_6, \mathtt{a}_7\}$;
  $\mathsf{Follow}_{\mathtt{r}_{\mathsf{Lin}}}(\mathtt{a}_7) = \{\mathtt{b}_8\}$; $\mathsf{Follow}_{\mathtt{r}_{\mathsf{Lin}}}(\mathtt{b}_8) = \{\mathtt{a}_6, \mathtt{a}_7\}$.

В описании данных множеств содержится исчерпывающая информация о языке $\mathscr{L}(\mathtt{r}_{\mathsf{Lin}})$.

Конструкция автомата Глушкова

Алгоритм построения $\mathtt{Glushkov}(r)$

- Строим линеаризованную версию r: $r_{\mathsf{Lin}} = \mathtt{Linearize}(r)$.
- Ищем $\mathsf{First}(r_{\mathsf{Lin}})$, $\mathsf{Last}(r_{\mathsf{Lin}})$ и $\mathsf{Follow}_{r_{\mathsf{Lin}}}(c)$ для всех $c \in \Sigma_{r_{\mathsf{Lin}}}$.
- Все состояния автомата, кроме начального (назовём его S), соответствуют буквам $c \in \Sigma_{r_{\mathsf{Lin}}}$.
- Из начального состояния строим переходы в те состояния, для которых $c \in \mathsf{First}(r_{\mathsf{Lin}})$. Переходы имеют вид $S \xrightarrow{c} c$.
- Переходы из состояния c соответствуют элементам d множества $\mathsf{Follow}_{r_{\mathsf{Lin}}}(c)$ и имеют вид $c \xrightarrow{d} d$.
- Конечные состояния — такие, что $c \in \mathsf{Last}(r_{\mathsf{Lin}})$, а также S, если $\varepsilon \in \mathscr{L}(R)$.
- Теперь стираем разметку, построенную линеаризацией, на переходах автомата. Конструкция завершена.

Пример автомата Глушкова

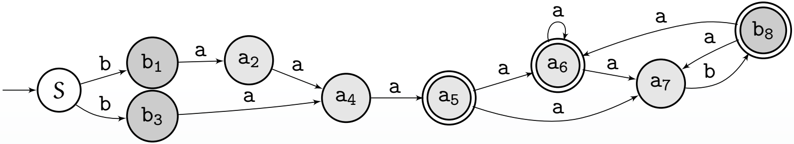
Исходное регулярное выражение:

$$(\mathtt{ba} \mid \mathtt{b})\mathtt{aa}(\mathtt{a} \mid \mathtt{ab})^*$$

Линеаризованное регулярное выражение:

$$(\mathtt{b}_1\mathtt{a}_2 \mid \mathtt{b}_3)\mathtt{a}_4\mathtt{a}_5(\mathtt{a}_6 \mid \mathtt{a}_7\mathtt{b}_8)^*$$

Автомат Глушкова:



Подграфы, распознающие регулярные выражения, являющиеся подструктурами исходного, не имеют общих вершин. Это свойство автомата Глушкова используется в реализациях `match`-функций некоторых библиотек регулярных выражений.

Свойства автомата Глушкова

- Не содержит ε-переходов.
- Число состояний равно длине регулярного выражения (без учёта регулярных операций), плюс один (стартовое состояние).
- В общем случае недетерминированный.

Примечание

Для 1-однозначных регулярных выражений $\mathtt{r}$ автомат $\mathtt{Glushkov}(\mathtt{r})$ является детерминированным. Эту его особенность активно используют в современных библиотеках регулярных выражений, например, в RE2. Выигрыш может получиться колоссальным: например, $\mathtt{Thompson}((a^*)^*)$ является экспоненциально неоднозначным, а $\mathtt{Glushkov}((a^*)^*)$ однозначен и детерминирован!

Бисимуляция

Labelled Transition Systems

Понятие бисимуляции возникло в контексте систем размеченных переходов (LTS).

Определение

Labelled Transition System — тройка $\langle S, \Sigma, Q \rangle$, где S — множество состояний, Σ — множество меток, Q — множество переходов (троек из $S \times \Sigma \times S$).

LTS похожи на конечные автоматы, но допускают бесконечные множества S и Q. Кроме того, в LTS нет начальных и финальных состояний.

Трансформационный моноид также строится в контексте LTS, то есть без учёта финальности состояний. Поэтому из ДКА, по которому строится трансформационный моноид, предварительно удаляются все ловушки, иначе в нём могут появиться правила переписывания, не имеющие никакого отношения к языку ДКА.

Симуляция и бисимуляция

Определение

Если $\precsim$ — симуляция для LTS = $\langle S, \Sigma, Q\rangle$, то

$$\forall p, q \in S(p \precsim q \Rightarrow (\exists p', a((p \xrightarrow{a} p') \Rightarrow \exists q'(q \xrightarrow{a} q' \;\&\; p' \precsim q'))$$

Если одновременно выполняются условия $p \precsim q$ и $q \precsim p$, то говорят, что p и q находятся в отношении бисимуляции (обозначается $p \sim q$).

Можно считать, что если $p \precsim q$, то множество путей в LTS, стартующих в p, вкладывается в множество путей с началом в q. Бисимуляция состояний в единственной LTS легко обобщается и на бисимуляцию между двумя разными LTS. Поскольку в них нет начальных состояний, и они не обязаны быть связными, можно рассматривать несколько LTS как одну LTS с несколькими компонентами и искать бисимуляцию между элементами этих компонент.

Бисимилярность НКА

Чтобы определить отношение бисимуляции на конечных автоматах, к отношению бисимуляции на LTS нужно добавить ограничения на бисимуляцию начальных и конечных состояний. Более точно, для бисимуляции НКА $\mathscr{A}_1$ и $\mathscr{A}_2$ необходимы следующие условия:

1. каждому состоянию $\mathscr{A}_1$ бисимилярно состояние $\mathscr{A}_2$, и наоборот;
2. стартовому состоянию $\mathscr{A}_1$ бисимилярно стартовое состояние $\mathscr{A}_2$;
3. каждому финальному состоянию $\mathscr{A}_1$ бисимилярно финальное состояние $\mathscr{A}_2$, и наоборот.

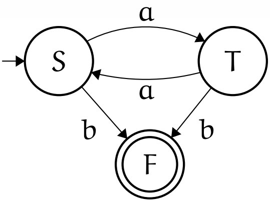
Лемма

Бисимилярные НКА распознают равные языки.

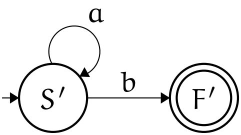
Пример бисимилярных НКА

Рассмотрим следующие два автомата, распознающие язык a^*b.

Автомат $\mathscr{A}_1$:

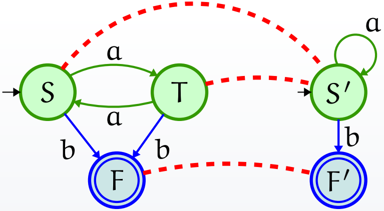


Автомат $\mathscr{A}_2$:



Их бисимуляция:

$$\{\langle S, S'\rangle, \langle T, S'\rangle, \langle F, F'\rangle\}$$



Состояния S и T бисимилярны одному и тому же состоянию S'.

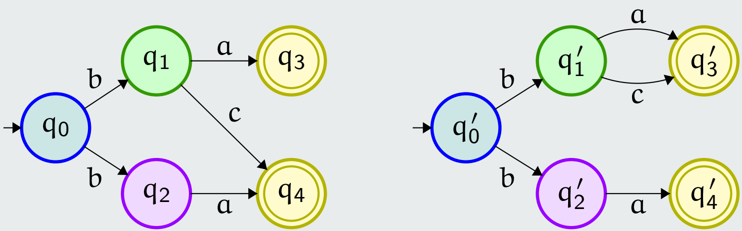
Бисимуляция и равенство

В равных НКА состояния бисимилярны, однако только условия существования бисимуляции и биекции бисимилярных состояний недостаточно, чтобы гарантировать равенство.

Пример неравных бисимилярных НКА

(автор примера: А. Д. Дельман)

Следующие два автомата бисимилярны и имеют одинаковое число состояний, однако не равны:



Бисимилярность состояний в НКА

Определение

Состояния q_i q_j в НКА $\mathscr{A}$ бисимилярны ($q_i \sim_{\mathscr{A}} q_j$), если они связаны LTS-бисимуляцией и имеют одинаковую финальность в $\mathscr{A}$.

- С учётом определения выше, бисимуляцию НКА можно переформулировать как отношение бисимуляции состояний НКА такое, что стартовые состояния бисимилярны.
- Отношение $\sim_{\mathscr{A}}$ имеет важное свойство: бисимилярные состояния в автомате можно объединить без изменения его семантики. Это преобразование часто позволяет существенно упростить НКА.

Бисимилярность состояний в НКА

Определение

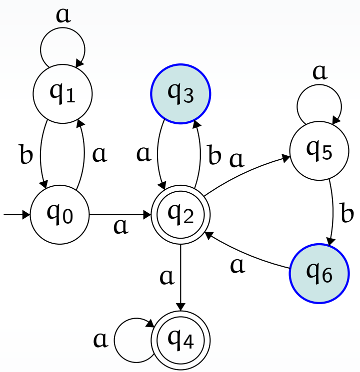
Состояния q_i q_j в НКА $\mathscr{A}$ бисимилярны ($q_i \sim_{\mathscr{A}} q_j$), если они связаны LTS-бисимуляцией и имеют одинаковую финальность в $\mathscr{A}$.

Пример

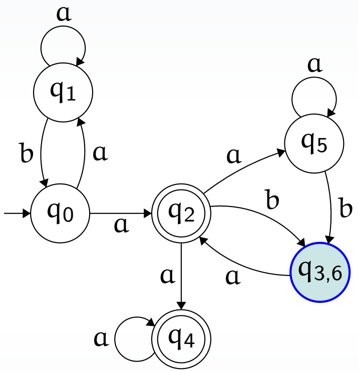
Все состояния-ловушки в любом полном автомате (т.е. с явно присутствующими переходами по всем буквам алфавита) бисимилярны друг другу. Все финальные состояния без переходов из них (кроме как в ловушки) также бисимилярны.

Пример слияния по бисимуляции

Исходный автомат:



Итоговый автомат:



Бисимуляция: $\left\{\{q_3, q_6\}, \bigcup\limits_{i\neq 3 \& i\neq 6} \{q_i\}\right\}$

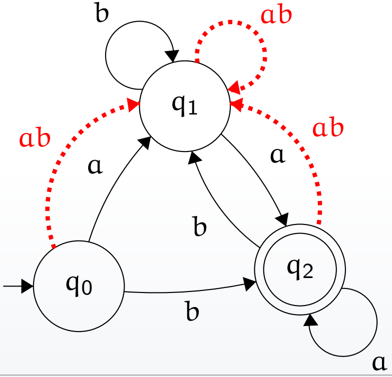
Кроме q_3 и q_6, все состояния не бисимилярны никаким другим.

Например, $q_1 \not\sim q_5$, поскольку $q_1 \xrightarrow{b} q_0$, $q_5 \xrightarrow{b} q_6$, но $q_0 \xrightarrow{a} q_1$ (и q_1 — не финальное), а из q_6 есть переход только в финальное состояние q_2.

Трансформационный моноид

Функции переходов по слову в ДКА

Правила перехода в ДКА $\mathscr{A}$ над алфавитом Σ и множеством состояний Q определяются функцией $\Sigma \times Q \to Q$. Если специализировать её по первому аргументу, получится функция $F_\xi : Q \to Q$ $(\xi \in \Sigma)$. Эту функцию можно продолжить на строки, положив $F_\xi \circ F_\eta = F_{\eta\xi}$.



Пусть мы строим функцию переходов по слову ab в автомате $\mathscr{A}$. Сначала определим функции F_a, F_b, определяющие его поведение на буквах a и b. Тогда поведение переходов на слове ab получится композицией F_a и F_b.

	q_0	q_1	q_2
a	q_1	q_2	q_2
b	q_2	q_1	q_1
ab	q_1	q_1	q_1

Функции переходов по слову в ДКА

Свойства множества функций переходов ДКА $\mathscr{A}$

- Существует единичная функция F_{ε} такая, что $\mathsf{F}_{\varepsilon} \circ \mathsf{F}_{\xi} = \mathsf{F}_{\xi} \circ \mathsf{F}_{\varepsilon} = \mathsf{F}_{\xi}$.
- Композиция $\circ$ ассоциативна.

Таким образом, функции переходов по словам из Σ^* в ДКА $\mathscr{A}$ образуют моноид относительно композиции.

Если ДКА представлен в краткой (trim) форме, некоторые переходы могут вести «в никуда». На самом деле они ведут в (единственное!) состояние-ловушку, существование которого неявно подразумевается. Однако наличие нескольких ловушек в ДКА повлечёт ошибки при построении функции переходов.

Определение и свойства

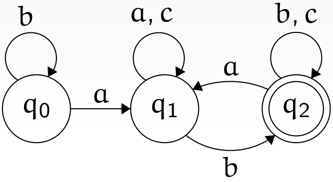
Определение

Трансформационный моноид $\mathcal{M}_{\mathscr{A}}$ для ДКА $\mathscr{A}$ — это моноид функций F_{ξ} таких, что $F_{\xi}(q_i) = q_j \Leftrightarrow (q_i \xrightarrow{\xi} q_j$ в $\mathscr{A})$. Иначе можно сказать, что трансформационный моноид $\mathcal{M}_{\mathscr{A}}$ определяется множеством классов эквивалентности $\{w \mid w \in \Sigma^{+}\}$ таким, что $w_i = w_j \Leftrightarrow F_{w_i} = F_{w_j}$.

Определение и свойства

- $\mathcal{M}_{\mathscr{A}}$ определяется фактормножеством классов эквивалентности и правилами переписывания, задающими эквивалентность. ε обычно не включается в множество w_i.
- Поскольку множество функций F_{w_i} в случае ДКА конечно, то $\mathcal{M}_{\mathscr{A}}$ содержит конечное число классов эквивалентности (верно и обратное: каждый такой моноид определяет ДКА).
- Трансмоноид строится для ДКА без ловушек; переход в ловушку обозначается в таблице переходов просто прочерком.
- Для единообразия записи трансформаций и перестановок в алгебре, в таблице переходов пишут только номера состояний $\mathscr{A}$.

Построение трансф. моноида

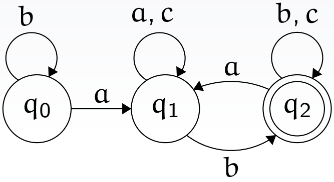


Определим соответствие между буквами и множествами переходов по ним и будем расширять этот список новыми словами в лексикографическом порядке. Если очередное слово задаёт такую же трансформацию, как и уже рассмотренное, порождаем соответствующее правило переписывания.

Классы эквивалентности	**Правила переписывания**

	0	1	2
a	1	1	1
b	0	2	2
c	—	1	2

Построение трансф. моноида



Классы эквивалентности

	0	1	2
a	1	1	1
b	0	2	2
c	—	1	2
ab	2	2	2
bc	—	2	2
ca	—	1	1

Правила переписывания

$aa \to a$	$ac \to a$
$ba \to a$	$bb \to b$
$cb \to bc$	$cc \to c$
$abc \to ab$	$bca \to ca$
$cab \to bc$	

Всего классов эквивалентности: 6

Синтаксический моноид

Определим отношение синтаксической конгруэнтности слов:

$$w_i \sim_{\mathscr{L}} w_j \Leftrightarrow \forall x, y (x\, w_i\, y \in \mathscr{L} \Leftrightarrow x\, w_j\, y \in \mathscr{L})$$

Синтаксический моноид $\mathcal{M}(\mathscr{L})$ — это множество его классов эквивалентности относительно $\sim_{\mathscr{L}}$. То есть такая полугруппа с единицей над $w \in \Sigma^*$, что $w_i = w_j \Leftrightarrow w_i \sim_{\mathscr{L}} w_j$ (равенство здесь понимается в алгебраическом смысле: как возможность преобразовать w_i и w_j к одному и тому же слову).

Лемма

Синтаксический моноид регулярного языка $\mathscr{L}$ совпадает с трансф. моноидом минимального ДКА, его распознающего.

Синтаксический моноид (так же, как и минимальный ДКА) — атрибут *языка*, а трансф. моноид — атрибут *конкретного ДКА*.

Суффиксная конгруэнтность

Предшествующие понятия рассматривали структуру переходов автомата без учёта начальных и конечных состояний, хотя неявно они использовались, чтобы удалить недостижимые состояния и состояния-ловушки при подготовке к построению моноида. Однако если чуть-чуть специализировать отношение $\sim_{\mathscr{L}}$, положив возможные префиксы пустыми, получится отношение эквивалентности, напрямую зависящее от положения стартовых и финальных состояний в минимальном ДКА.

Определим отношение эквивалентности по Нероуду как:

$$w_i \equiv_{\mathscr{L}} w_j \Leftrightarrow \forall y (w_i\, y \in \mathscr{L} \Leftrightarrow w_j\, y \in \mathscr{L})$$

Обозначение $\sim_{\mathscr{L}}$ может использоваться в литературе как в смысле синтаксической конгруэнции, так и в смысле эквивалентности по Нероуду. Лучше дополнительно уточнить.

Критерий регулярности языка

Теорема Майхилла–Нероуда

Язык $\mathscr{L}$ регулярен тогда и только тогда, когда множество классов эквивалентности по $\equiv_{\mathscr{L}}$ конечно.

$\Rightarrow$: Пусть $\mathscr{L}$ регулярен. Тогда он порождается некоторым DFA $\mathscr{A}$ с конечным числом состояний N. Значит, множество $\{q_i \mid q_0 \xrightarrow{w} q_i\}$ конечно, а для каждых двух w_1, w_2 таких, что $q_0 \xrightarrow{w_1} q_i$ и $q_0 \xrightarrow{w_2} q_i$, выполняется $w_1 \equiv_{\mathscr{L}} w_2$.

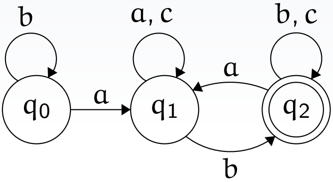
$\Leftarrow$: Пусть все слова в Σ^* принадлежат N классам эквивалентности $A_1, \ldots, A_n$ по $\equiv_{\mathscr{L}}$. Построим по ним DFA $\mathscr{A}$, распознающий $\mathscr{L}$. Классы A_i сопоставим состояниям:

- Начальным объявим класс эквивалентности A_0 такой, что $\varepsilon \in A_0$.
- Конечными объявим такие A_j, что $\forall w \in A_j (w \in \mathscr{L})$.
- Если $w \in A_i$, $w a_k \in A_j$, тогда добавляем в δ правило $\langle A_i, a_k, A_j \rangle$.

Трансформационный моноид и $\equiv_{\mathscr{L}}$

- Классы эквивалентности по Майхиллу–Нероуду можно извлечь из трансформационного моноида минимального ДКА для языка $\mathscr{L}$: они являются подмножеством факторслов, которые переводят стартовое состояние в какое-то другое (непустое) состояние.
- Для каждой пары таких факторслов w_i и w_j, переводящих стартовое состояние в разные состояния q_i и q_j, в синтаксическом моноиде обязательно найдётся различающий суффикс (т.е. класс эквивалентности u такой, что $w_i u \in \mathscr{L}$ & $w_j u \notin \mathscr{L}$, либо наоборот).
- Если в ДКА существует ловушка (возможно, неявная), то в трансформационном моноиде найдётся класс эквивалентности, переводящий в неё стартовое состояние. Таких классов может быть несколько, но с точки зрения эквивалентности по Майхиллу–Нероуду, они не различаются.

Пример



Включим в число факторслов ε и выделим по одному факторслову для каждого состояния q_i, переводящему стартовое слово в q_i. Для каждого из них определим множество суффиксов, которые оставляют слова этих классов в языке. После этого достаточно собрать вместе все суффиксы и префиксы и выкинуть из полученной таблицы дубли столбцов.

Факторслова–префиксы

префикс	$0 \to ?$	суффиксы
ε	0	ab
a	1	b, ab
ab	2	ε, b, ab

Таблица классов эквивалентности

	ε	b	ab
ε	−	−	+
a	−	+	+
ab	+	+	+