конспекты занятий по курсу

«Теория и реализация языков программирования»

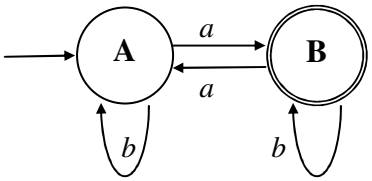
Занятие 4

4. Как найти регулярный язык, дополнительный к исходному, как минимизировать конечный автомат или для чего КА выворачивают наизнанку…

Это занятие посвящено двум разным приёмам обращения с КА, которые предполагают одни и те же допущения относительно обрабатываемого КА. Иными словами для правильной работы обоих алгоритмов КА на входе должен удовлетворять определённым требованиям, с необходимостью которых мы вначале и познакомимся на примерах.

4.1. Как найти регулярный язык, дополнительный к исходному

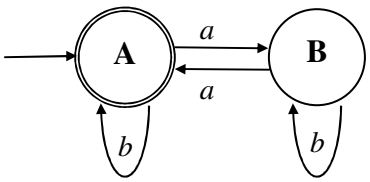
Пусть язык L_1 задан следующим автоматом



$$L_1 = \{x \in \{a,b\}^* : |x|_a - \text{нечётно}\}$$

Преобразуем этот КА следующим образом: оставив прежние дуги и состояния, заменим каждое заключительное состояние исходного автомата на незаключительное и наоборот – все исходные незаключительные состояния сделаем заключительными.

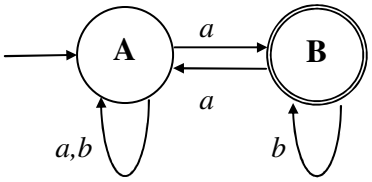
Станет ли новый автомат задавать язык, дополнительный тому, который задавался исходным КА?



$$L_2 = \{x \in \{a,b\}^* : |x|_a - \text{чётно}\}$$

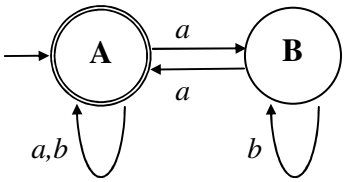
Да, это так. Очевидно, $L_1 = \overline{L_2}$.

Попробуем выполнить тот же приём для похожего автомата.



$$L_3 = \{x \in \{a,b\}^* : |x|_a > 0\}$$

Станет ли новый автомат задавать язык, дополнительный тому, который задавался исходным КА?



$L_4 = \{x \in \{a,b\}^*\}$

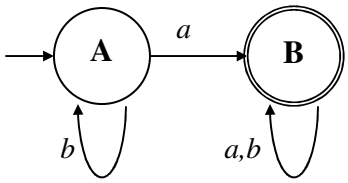
Видим, что новый автомат задаёт не только цепочки из дополнения языка L_3, т. е. $\overline{L_3} = \{x \in \{a,b\}^* : |x|_a = 0\}$, но и вообще все возможные цепочки над алфавитом {*a, b*}.

Поскольку автомат для L_3, очевидно, неопределённен, разумно предположить, что это одна из причин, по которой и произошло данное отклонение. М.б. устранение неопределённости исходного КА поможет правильно выполниться и алгоритму обращения автомата?

Проверим это. Переведём НКА для L_3 в ДКА (см. предыдущие занятия)

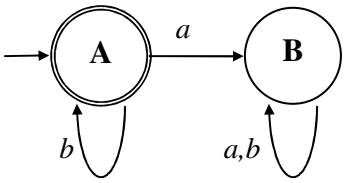
	a	*b*
A	AB	A
AB	AB	AB

Переобозначив новые состояния как A и B, получим следующую диаграмму ДКА:



$L_3 = \{x \in \{a,b\}^* : |x|_a > 0\}$

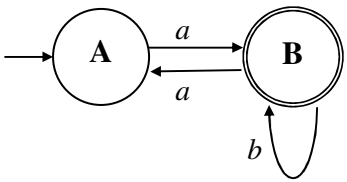
После обращения автомата, представленного в данном виде, получим:



Получили $L_5 = \{x \in \{a,b\}^* : |x|_a = 0\} = \overline{L_3}$.

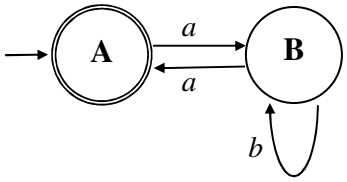
Т.е. требование определённости КА, похоже, является необходимым для успешного применения обращения КА для нахождения дополнения регулярного языка. А является ли оно достаточным?

Рассмотрим ещё один пример:



$$L_6 = a(b^* aa\ b^*)^*$$

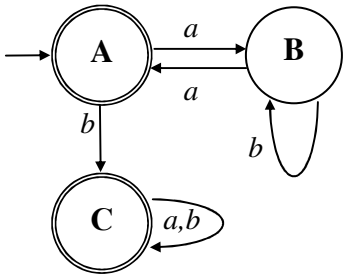
После обращения данного ДКА получим:



$$L_7 = (ab^* a)^*$$

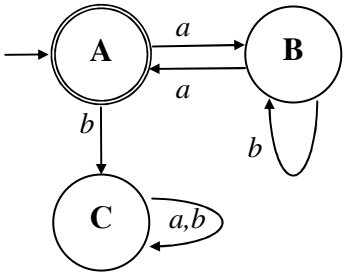
В новом ДКА все цепочки принадлежат дополнению исходного языка, но часть из цепочек из дополнения них при этом явно пропущена. Так, нет ни одной цепочки, которая начиналась бы на знак «*b*».

В чём же дело? Попробуем достроить полученный ДКА так, чтобы он принимал также и все цепочки, начинающиеся на «*b*»



$$L_8 = (ab^* a)^* \mid b(a \mid b)^* = \overline{L}_6$$

… и применить алгоритм обращения в обратном порядке, чтобы увидеть, что же было упущено в исходном ДКА:



$$L_6 = a(b^* aa\ b^*)^*$$

Да, этот автомат задаёт язык L_6, но ведь состояние C – явно «лишнее», поскольку оно не позволяет определить ни одной цепочки языка! Да, это так. Но оно позволяет сделать данный автомат полностью определённым, т.е. из каждого состояния такого автомата есть дуги по всем знакам основного алфавита языка. Именно это свойство КА позволяет успешно применять к ним ряд алгоритмов

преобразования, ещё один из которых мы рассмотрим чуть ниже.

А пока в качестве отдыха позволим на минутку отвлечь внимания уважаемого читателя замечанием о возможном философском истолковании свойства полной определённости. Состояние «С» напоминает мне о «спящих», не востребованных в обычной жизни возможностях. Это можно понять и как возможности ума и тела отдельного человека, и как возможности страны или всего человечества. Они спят до времени, пока под воздействием перемен и вызовов времени не проявится и не будет настоятельно востребована в них необходимость. И тогда спящий проснётся…

4.2. Всюду определённые (полные) ДКА

Дадим формальное определение всюду определённого (полного) ДКА и алгоритм его построения.

Под полным ДКА подразумевается ДКА, у которого

$\delta(A,a) \neq \varnothing \quad \forall \quad A \in Q \quad$ и $\quad \forall\, a \in T.$

Доказано ([1]), что полный ДКА всегда может быть получен из неполного следующим образом:

а. Добавляем к неполному ДКА новое незавершающее состояние Z.

б. Функции перехода полного ДКА получаем следующим образом

1) $\forall\, A \in Q$ и $\forall\, a \in T,$ которым соответствует $\delta(A,a) \neq \varnothing$

сохраняем старую функцию переходов: δ'(A, a) = δ(A, a).

2) $\forall\, A \in Q$ и $\forall\, a \in T,$ которым соответствует $\delta(A,a) = \varnothing$

добавляем функцию перехода в новое состояние: δ'(A, a) = Z.

3) $\forall\, a \in T$ добавляем функцию перехода из нового состояния Z в него же:

δ' (Z, a) = Z $\quad (\forall\, a \in T)$.

Пример. Именно этот алгоритм мы чуть выше применили к ДКА, задающего L_7, чтобы получить из него всюду определённый ДКА (диаграмма для языка L_8).

4.3. Алгоритм минимизации полного ДКА

Возможность минимизировать ДКА интересна:

- с точки зрения технических приложений (ведь при воплощении в технике, например, в соответствующей микросхеме, каждое не являющееся необходимым состояние КА в общем случае требует дополнительных материалов, места, энергии…)
- с научной и учебной точки зрения, ведь это один из удобных способов сравнить два по-разному представленных ДКА и убедиться (или опровергнуть), что они задают один и тот же регулярный язык.

Алгоритм минимизации полного ДКА состоит в следующем [1]:

(1) Построить начальное разбиение множества состояний из двух групп:

заключительные состояния {F} и остальные {Q-F}.

(2) Для каждой имеющейся группы G разбить G на подгруппы так, чтобы состояния *s* и *t* из G оказались в одной подгруппе тогда и только тогда, когда для каждого входного символа *a* состояния *s* и *t* имеют переходы по *a* в состояния из одной и той же ранее найденной нами группы;

заменить G на множество всех полученных подгрупп;

(3) Если при выполнении (2) не удалось разбить ни одной группы, переходим к шагу 4,

иначе повторяем шаг 2.

(4) Определим новые состояния ДКА, как получившиеся группы состояний минимизируемого автомата.

Если группа содержит начальное состояние автомата M, то эта группа становится начальным состоянием автомата M'. Если группа содержит заключительное состояние M, она становится заключительным состоянием M'.

Отметим, что каждая полученная группа либо состоит только из состояний из {F}, либо не имеет состояний из {F}. Переходы определяются очевидным образом.

(5) Если M' имеет «мёртвое» состояние, то есть состояние, которое не является допускающим и из которого нет путей в допускающие, удалить его и связанные с ним переходы из M'.

Удалить из M' также все состояния, недостижимые из начального.

Пример 4.1. Минимизация ДКА для языка $\{x \in \{a, b\}^* : |x|_a$ – чётно$\}$

Как мы успели заметить, ДКА, полученный по алгоритму из НКА, соответствующий РВ (ab*a | b)*, явно имеет избыточные состояния.

Поскольку этот ДКА оказался полным, к нему можно применить алгоритм минимизации полного ДКА:

(1) Построим начальное разбиение множества состояний из двух групп:

заключительные состояния F = {A, C, D}

и остальные Q-F = {B, E}.

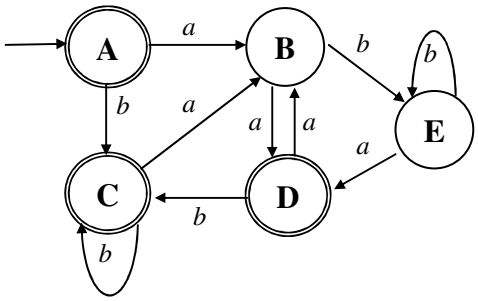


Рис. 4.1. ДКА, полученный по алгоритму из НКА, соответствующий РВ (ab*a | b)*

(2-3). Выполнение шагов (2) и (3) алгоритма удобно отслеживать на диаграмме, по горизонтальной оси которой выписывается перечень всех вершин ДКА от первой до предпоследней, а по вертикальной (сверху вниз) – от второй до последней.

B	×			
C	0	×		
D	0	×	0	
E	×	0	×	×
	A	B	C	D

Рис. 4.2. Диаграмма определения равнозначных (эквивалентных) состояний для минимизируемого ДКА.

Знаком × здесь обозначены неравнозначность состояний, знаком 0 – равнозначность.

Первый раз они проставляются так.

Пусть у нас обе группы состояний не пусты, т.е. $F \neq \varnothing$ и $Q - F \neq \varnothing$ (иначе имеем вырожденные случаи, соответственно, с ничего не принимающим ДКА и, во втором случае, с ДКА, все состояния которого равнозначны, т.е. ДКА с единственным состоянием, которое является и входным и заключительным).

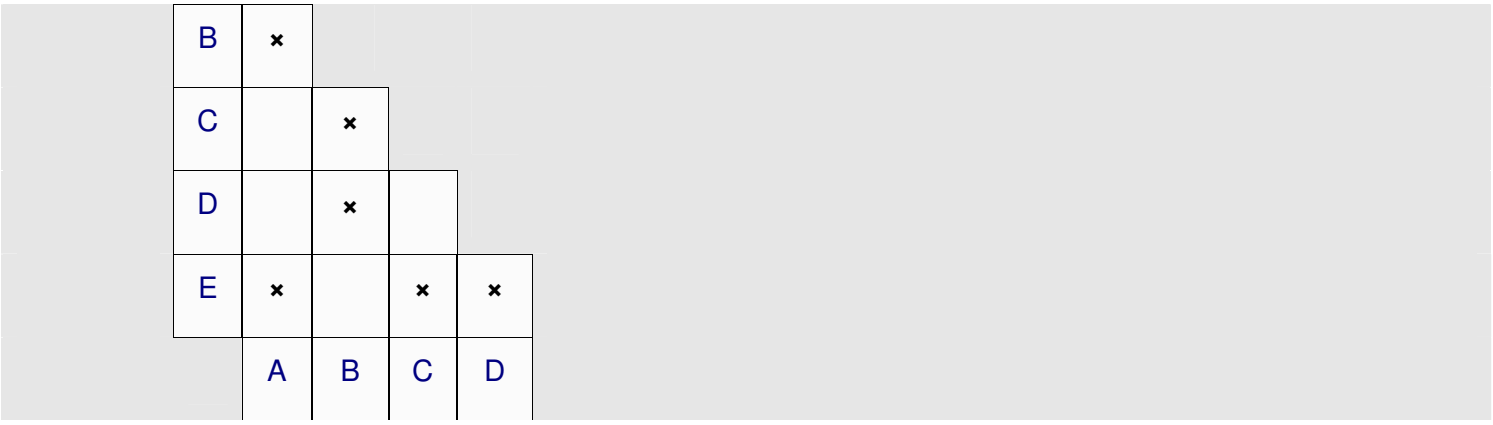
Тогда на пересечении каждого состояния из F с каждым состоянием из *Q - F* ставим × (поскольку они уже принадлежат к разным группам состояний).

В нашем примере

F = {A, C, D}, a

Q–F = {B, E},

поэтому, скажем, состояния A и B, A и E изначально являются неэквивалентными:



После этого к наметившимся кандидатам в равнозначные состояния (C и A, D и A, D и C, E и B) применяем шаг 2 Алгоритма минизации до тех пор, пока удаётся получать новые разбиения состояний.

Например, для пары C и A, проверяем – нет ли переходов из них по знаку «a» в разные группы состояний.

δ**(A, a) = B**

δ **(C, a) = B**

По знаку «a» попадаем не только в одну группу состояний, но даже в одно и тоже состояние, разницы нет.

Проверяем переходы по знаку «b»:

δ **(A, b) = C**

δ **(C, b) = C**

Также нет разницы.

Тот же итог для данного примера имеем и для других пар. Разбиение закончено.

(4) Определим новые состояния ДКА, как получившиеся группы состояний минимизируемого автомата.

$$A' = \{A, C, D\}$$

$$B' = \{B, E\}$$

Минимизированный автомат будет иметь вид

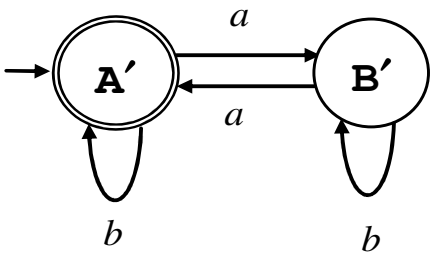
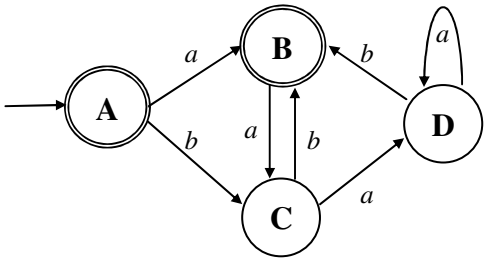


Рис. 4.3. Минимизированный ДКА.

Задача 4.1. Исследуйте возможность минимизации следующего ДКА:



4.3. Примеры задач на регулярные языки

Пример 4.2. Построить ДКА для языка {x ∈ {0, 1}* : на каждом кратном двум месте цепочки которого (1, 3, 5, …) стоит знак 0 (нумерация знаков начинается с 1), а $|x|_1$ – чётно}.

Самые короткие цепочки из данного языка такие:

ε , 0, 00, 000, 0000, 0101 и т.д.

Не будем торопиться переходить к решению этой, в общем-то, простой задачи, а сначала постараемся разобрать её условие, понять, что оно может нам подсказать о будущих свойствах ДКА или ПГ, которые соответствуют данному языку.

Заметим, что в условии задачи указаны два ограничения, одно из которых касается нулей, второе – единиц.

По нулям имеем следующие состояния:

A – мы на кратном двум месте цепочки (начиная с 1-го), здесь должен стоять 0

B – мы на некратном двум месте, здесь может стоять как 0, так и 1.

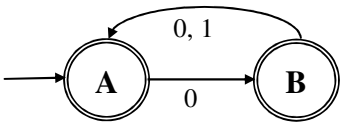


Рис. 4.4. ДКА для подмножества искомого языка {x ∈ {0, 1}* : на каждом кратном двум месте цепочки которого (1, 3, 5, …) стоит знак 0 (нумерация знаков начинается с 1).

По единицам состояния таковы:

C – в данное время принято (порождено) чётное число единиц, это состояние может быть завершающим.

D - $|x|_1$ – нечётно, состояние не может быть завершающим.

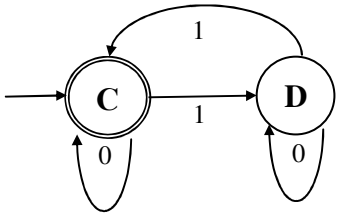


Рис. 4.5. ДКА для подмножества искомого языка {x ∈ {0, 1}* : $|x|_1$ – чётно).

Учёт каждого из этих ограничений по отдельности требует самое меньшее двух состояний. Поскольку ни одно из ограничений не зависит от другого, то комбинаторика подсказывает, что искомый ДКА должен содержать хотя бы $2 \times 2 = 4$ состояния.

Попробуем их перечислить:

A - кратное двум место цепочки, $|x|_1$ – чётно. Состояние завершающее.

B - некратное двум место цепочки, $|x|_1$ – чётно. Состояние завершающее

C - кратное двум место цепочки, $|x|_1$ – нечётно. Состояние не завершающее

D - некратное двум место цепочки, $|x|_1$ – нечётно. Состояние не завершающее

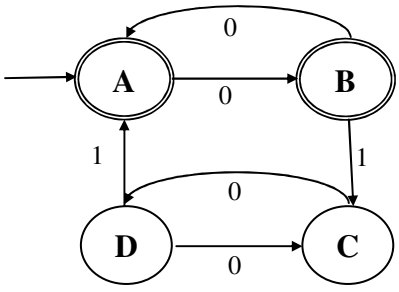


Рис. 4.6. ДКА – итог перемножения двух частных ДКА - подмножеств искомого языка.

Задача 4.2. Построить ДКА для языка {x ∈ {0, 1}* : на каждом кратном трём месте цепочки

которого (1, 4, 7, …) стоит знак 0 (нумерация знаков начинается с 1), а $|x|_1$ – нечётно}. Исследовать полученный ДКА на минимальность состояний.

Совет. Познакомьтесь с более кратким математическим описанием соответствующих алгоритмов в пособии [1] и введёнными при этом понятиями и обозначениями.

== Ссылки ==

[1] *Серебряков В.А.* [и др.]. Теория и реализация языков программирования. М.: МЗ-Пресс, 2003 (1-е изд.) и 2006 (2-е изд). – 294 с. (681.3/ Т33). http://trpl7.ru/t-books/TRYAP_BOOK_Details.htm