Минимазация детерминированного конечного автомата.

Цель работы: Оценка ассимптотической сложности алгоритмов. Исследование алгоритмов минизации детерминированного конечного автомата.

Описание работы

Задан детерминированный конечный автомат (ДКА). Требуется построить эквивалентный ему автомат с минимальным количеством состояний.

Проверка эквивалентности состояний

Состояния p и q эквивалентны, если для всех входных цепочек w состояние $\hat{\delta}(p,w)$ является допускающим тогда и только тогда, когда состояние $\hat{\delta}(q,w)$ – допускающее.

Менее формально, эквивалентные состояния p и q невозможно разлечить, если просто прооверить, допускает ли автомат данную входную цепочку, начиная работу в одном (неизвестно, каком именно) из этих состояний. Заметим, что состояния $\hat{\delta}(p,w)$ и $\hat{\delta}(q,w)$ могут и не совпадать – лишь бы оба были либо допускающими, либо недопускающими.

Если два состояния p и q не эквивалентны друг другу, то будем говорить, что они различимы, т.е. существует хотя бы одна цепочка w, для которой одно из состояний $\hat{\delta}(p,w)$ и $\hat{\delta}(q,w)$ является допускающим, а другое – нет.

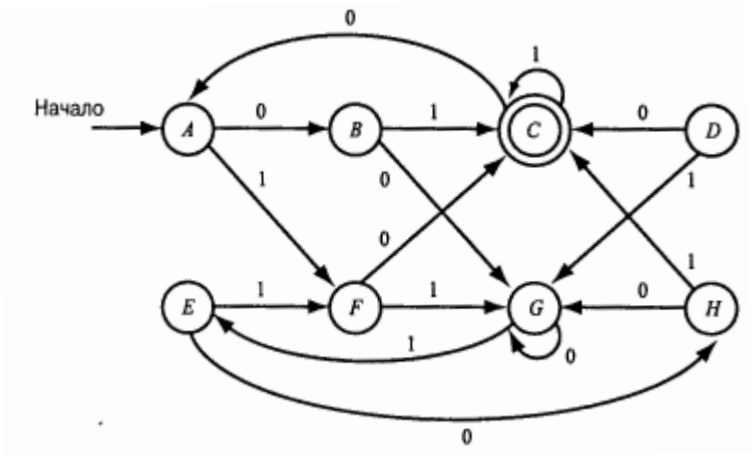

Рис.1. Автомат с эквивалентными состояниями.

Рассмотрим ДКА на рис. 1. Очевидно, что некоторые пары состояний не эквивалентны, например C и G, потому что первое из них допускающее, а второе – нет. Пустая цепочка различает эти состояния, так как $\hat{\delta}(\mathrm{C},\varepsilon)$ - допускающее состояние, а $\hat{\delta}(G,\varepsilon)$ – нет.

Рассмотрим состояния A и G. Различить их с помощью цепочки ε невозможно, так как они оба не допускающие. 0 также не различает их, поскольку по входу 0 автомат переходит в состояния B и G, соответственно, а оба эти состояния недопускающие. Однако цепочка 01 различает A и G, так как $\hat{\delta}(A,01)=C$, $\hat{\delta}(G,01)=E$, состояние C – допускающее, а E - нет. Для доказательства неэквивалентности A и G достаточно любой входной цепочки, переводящие автомат из состояний A и G в состояния, одно из которых является допускающим, а второе – нет.

Для того, чтобы найти эквивалентные состояния, нужно выявить все пары различимых состояний. Как ни странно, но если найти все пары состояний, различимых в соответвии с представленном ниже алгоритмом, то те пары состояний, которые найти не удастся, будут эквивалентными. Алгоритм, который называется алгоритмом заполнения таблицы, состоит в рекурсивном обнружении пар различимых состояний ДКА.

Базис. Если состояние p – допускающее, а q – недопускающее, то пара состояний $\{p, q\}$ – различима.

Индукция. Пусть p и q – состояния, для которых существует входной символ a, приводящий из в различимые состояния $r = \delta(p, a)$ и $s = \delta(q, a)$. Тогда $\{p, q\}$ – пара различимых состояний. Это правило очевидно, потому что должна существовать цепочка w, отличающая r и s, т.е. только одно из состояний $\hat{\delta}(r, w)$ и $\hat{\delta}(s, w)$ является допускающим. Тогда цепочка aw отличает p от q, так как $\hat{\delta}(p, aw)$ и $\hat{\delta}(q, aw)$ – это та же пара состояний, что и $\hat{\delta}(r, w)$ и $\hat{\delta}(s, w)$.

Выполним алгоритм заполнения тблицы для ДКА, представленного на рис. 1. Окончательный вариант таблицы изображен на рис. 2, где x обозначает пары различимых состояний, а пустые ячейки указывают пары эквивалентных состояний. Сначала в таблице нет ни одного x.

В базисном случаи, поскольку C – единственное допускающее состояние, записываем x в каждую пару состояний, в которую входит C. Зная некоторые пары различимых состояний, можно найти другие. Например, поскольку пара $\{C, H\}$ различима, а состояния E и F по входу 0 переходят в H и C, соответственно, то пара $\{E, F\}$ также неразличима. Фактически, все x на рис. 2, за исключением пары $\{A, G\}$, получаются очень просто: посмотрев на переходы из каждой пары состояний по символам 0 и 1, обнаружим (для одного из этих символов), что одно состояние переходит в C, а другое – нет. Различимость пары состояний $\{A, G\}$ видна в следующем цикле, поскольку по символу 1 они переходят в F и E, соответсвенно, а различимость состояний $\{E, F\}$ уже установлена.

	A	B	C	D	E	F	G
B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x

Рис. 2. Таблица неэквивалентности состояний.

Однако обнаружить другие пары различимых состояний невозможно. Следовательно, оставшиеся пары состояний $\{A, E\}, \{B, H\}$ и $\{D, F\}$ эквивалентны. Выясним, почему нельзя утверждать, что пара состояний $\{A, E\}$ различима. По входному символу 0 состояния A и E переходят в состояния B и H, соответственно, а про эту пару пока неизветно, различима она, или нет. По символу 1 оба состояния A и E переходят в F, так что нет никакой надежды различить их этим способом. Остальные две пары, $\{B, H\}$ и $\{D, F\}$, различить нельзя, поскольку у них одинаковые переходы как по символу 0, так и по 1. Таким образом, алгоритм заполнения таблицы останавливается на таблице представленной на рис. 2, и корректно определяет эквивалентные и различимые состояния.

Теорема

Если два состояния не различаются с помощью алгоритма заполнения таблицы, то они эквивалентны.

На рис. 3 изображен ДКА с минимильным числом состояний, эквивалентный автомату, представленному на рис. 1.

Минимизаци ДКА за $O(n^2)$

Основная идея алгоритмов минимизации ДКА заключается в том, чтобы разбить состояния на классы эквивалентности – они и будут состояниями минимизированного

автомата. Существует несколько алгоримов, которые позволяют разбить состояния на классы эквивалентности.

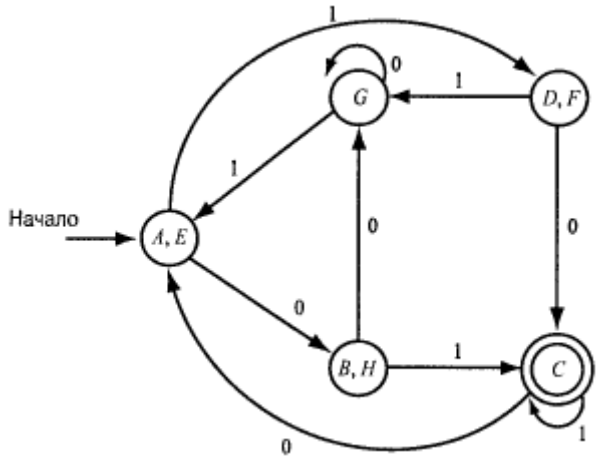

Рис. 3. ДКА с минимальным числом состояний.

Рассмотрим первый алгоритм минимизации ДКА, работающий за $O(n^2)$. Для реализации алгоритма нам потребуются очередь Q и таблица размером $n \times n$, где n – количество состояний автомата.

Будем помечать в таблице пары неэквивалентных состояний и класть их в очередь.

Изначально добавим в очередь Q пары состояний, различимых строкой ε, и пометим их в таблице. Пока Q не станет пуста, будем делать следующее:

1. извлечем пару (u, v) из Q;
2. отметим в таблице и добавим в очередь Q все пары (t, k) такие, что $\exists c \in \Sigma$, $\delta(t, c) = u$, $\delta(k, c) = v$, и пара (t, k) не отмечена в таблице.

В момент опустошения очереди пары состояний, не помеченные в таблице, являются парами эквивалентных состояний. За один проход по таблице разбиваем пары эквивалентных состояний на классы эквивалентности.

Стартовым состоянием полученного автомата будет состояние, соответствующее классу эквивалентности, содержащему стартовое состояние исходного автомата.

Терминальными состояниями полученного автомата будут состояния, соответствующие классам эквивалентности, содержащим терминальные состояния исходного автомата.

Для втрого алгоритма минимизации ДКА необходимо ввести дополнительное определение.

Класс S разбивает класс R по символу a на R_1 и R_2, если

1. $\forall r \in R_1$ $\delta(r, a) \in S$;
2. $\forall r \in R_2$ $\delta(r, a) \notin S$.

Если класс R может быть разбит по символу a, то он содержит хотя бы одну пару неэквивалентных состояний (так как существует строка которая их различает). Если класс нельзя разбить, то он состоит из эквивалентных состояний. Поэтому самый простой алгоритм состоит в том, чтобы разбивать классы текущего разбиения до тех пор пока это возможно.

Итеративно строим разбиение множества состояний следующим образом.

1. Первоначальное разбиение множества состояний – класс допускающих состояний Q и класс недопускающих состояний $Q \backslash F$.
2. Перебираются символы алфавита $a \in \Sigma$, все пары (Q, a) и $(Q \backslash F, a)$ помещаются в очередь.
3. Из очереди извлекается пара (S, a), Sдалее именуется как сплиттер.

4. Все классы текущего разбиения разбиваются на 2 подкласса (один из которых может быть пустым). Первый состоит из состояний, которые по символу a переходят в сплиттер, а второй из всех оставшихся.
5. Те классы, которые разбились на два непустых подкласса, заменяются этими подклассами в разбиении, а также добавляются в очередь.
6. Пока очередь не пуста, выполняем п.3 – п.5.

Псевдокод

Q – множество состояний ДКА. F – множество терминальных состояний. W –очередь. P – разбиение множества состояний ДКА. R – класс состояний ДКА.

```
P ← {F, Q\F}
W ← {}
for all a ∈ Σ
   W.push(F, a)
   W.push(Q\F, a)
while not W.isEmpty()
   W.pop(S, a)
   for all R in P
      R1 = R ∩ δ^-1(S, a)
      R2 = R\R1
      if |R1| ≠ 0 and |R2| ≠ 0
         replace R in P with R1 and R2
         W.push(R1)
         W.push(R2)
```

Когда очередь станет пустой, будет получено разбиение на классы эквивалентности, так как больше ни один класс невозможно разбить.

Время работы алгоритма оценивается как $O(|\Sigma| \cdot n^2)$, где n – количество состояний ДКА, а Σ – алфавит. Это следует из того, что если пара (S, a) попала в очередь, и класс S использовался в качестве сплиттера, то при последующем разбиении этого класса в очередь добавляется два класса S_1 и S_2, причем можно гарантировать лишь следующее уменьшение размера: $|S| \geq |S_i| + 1$. Каждое состояние изначально принадлежит лишь одному классу в очереди, поэтому каждый переход в автомате будет просмотрен не более, чем $O(n)$ раз. Учитывая, что ребер всего $O(|\Sigma| \cdot n)$, получаем указанную оценку.

Минимизаци ДКА за $O(n\log n)$

Алгоритм минимазации ДКА за время $O(n\log n)$ называется алгоритмом Хопкрофта.

Лемма

Класс $R = R_1 \cup R_2$ и $R_1 \cap R_2 = \emptyset$, тогда разбиение всех классов (текущее разбиение) по символу a любыми двумя классами из R, R_1, R_2 эквивалентно разбиению всех классов с помощью R, R_1, R_2 по символу a.

Алгоритм Хопкрофта отличается от простого тем, что иначе добавляет классы в очередь. Если класс R уже есть в очереди, то согласно лемме можно просто заменить его на R_1 и R_2. Если класса R нет в очереди, то согласно лемме в очередь можно добавить класс R и любой из R_1 и R_2, а так как для любого класса B из текущего разбиения выполняется

$$\forall r \in B\ \delta(r, a) \in S \text{ or } \forall r \in B\ \delta(r, a) \notin S,$$

то в очередь можно добавить только меньшее из R_1 и R_2.

Псевдокод

Q – множество состояний ДКА. F – множество терминальных состояний. W –очередь. P – разбиение множества состояний ДКА. R – класс состояний ДКА.

```
P ← {F, Q\F}
W ← {}
if |F| ≤ |Q\F|
  for all a ∈ Σ
    W.push(F, a)
else
  for all a ∈ Σ
    W.push(Q\F, a)
while not W.isEmpty()
  W.pop(S, a)
  for all R in P split by S
    replace R in P with R1 and R2
    if (R, a) in W
      replace (R, a) with (R1, a) and (R2, a)
    else
      if |R1| ≤ |R2|
        W.push(R1)
      else
        W.push(R2)
```

Время работы модифицированного алгоритма оценивается как $O(|\Sigma| \cdot n\log n)$, где n – количество состояний ДКА, а Σ – алфавит. В данном случае при последующем разбиении в очередь будет добавлен класс S_1, причем $|S| \geq 2|S_1|$. Каждый переход в автомате будет просмотрен не более, чем $O(\log n)$ раз, ребер всего $O(|\Sigma| \cdot n)$, отсюда указанная оценка.

Формат входного файла

Во первой строке входного файла содержатся числа n, m и k – количество состояний, переходов и допускающих состояний в автомате соответственно. $(1 \leq n, m \leq 1000, 1 \leq k \leq n)$.

В следующей строке содержатся k чисел – номера допускающих состояний (состояния пронумерованы от 1 до n).

В следующих m строках описываются переходы в формате "$a\ b\ c$", где a – номер исходного состояния перехода, b – номер состояния, в которое осуществляется переход и c - символ (строчная латинская буква), по которому осуществляется переход.

Стартовое состояние автомата всегда имеет номер 1. Гарантируется, что из любого состояния не более одного перехода по каждому символу.

Формат выходного файла

Требуется выдать результирующий автомат в том же формате.

Пример

minimization.in	minimization.out
2 2 2 1 2 1 2 a 2 2 a	1 1 1 1 1 1 a

Порядок выполнения работы.

1. Изучение теоретического материала: ассимптотические оценки алгоритмов, алгоритмы минимизации ДКА.
2. Реализовать алгоритмы минимизации ДКА работающие за $O(n^2)$ и за $O(n\log n)$.
3. Реализовать алгоритм для создания тестовых входных файлов.
4. Исследовать быстродействие алгоритмов минимизации ДКА.
5. Оформление отчета по лабораторной работе.
6. Защита лабораторной работы.

Содержание отчета.

1. Титульный лист.
2. Цели и задачи лабораторной работы.
3. Описание алгоритмов, использованных в лабораторной работе.
4. Результаты исследование быстродействия рассмотренных алгоритмов (график зависимости времени работы алгоритма от размера входных данных).
5. Описание программы: язык программирования, среда разработки, ограничение на входных/выходные данные, формат входных данных, особенности реализации.
6. Пример работы программы на различных входных данных.
7. Исходный код программы. В отчет можно включать не полный листинг программы, а лишь алгоритмически или логически важные части программы.

Литература

1. Хопкрофт Д., Мотвани Р., Ульман Д. Введение в теорию автоматов, языков и вычислений, 2-е издание. Пер. с англ. – М.: Издательский дом «Вильямс», 2002. – С. 171 - 182.
2. *D. Gries.* Describing an algorithm by Hopcroft. Technical Report TR-72-151, Cornell University, December 1972.