Детерминизация конечного автомата

Кузнецов А.Д.

Оглавление

0.1 Краткая теоретическая часть

Ниже приводятся определения и утверждения, которые будут использованы для дальнейших пояснений к реализации алгоритма детерминизации.

0.1.1 Автомат

Конечный недетерминированный автомат (КНА) M — это кортеж вида

$$M = (A, Q, q_0, F, \delta),$$

где

- $A = \{a_0, a_1, \ldots, a_{m-1}\}$ — входной алфавит, т.е. множество символов, причем конечное : $|A| = m$;
- $Q = \{q_0, q_1, \ldots, q_{n-1}\}$ — множество состояний автомата, тоже конечное: $|Q| = n$;
- q_0 — начальное состояние автомата, то есть $q_0 \in Q$;
- $F \subset Q$ — выходные состояния автомата;
- $\delta : Q \times A \to 2^Q$ — функция переходов автомата.

Если в автомате разрешены переходы по пустому символу, то входной алфавит дополняется фиктивным символом λ (в иностранной литературе его чаще обозначают ε). В таком случае автомат называют λ-КНА (ε-FSA). Входной алфавит при этом будем обозначать так же, то есть если речь идет о λ-КНА, то подразумевается, что $A \leftarrow A \bigcup \{\lambda\}$, а мощность $|A| = m + 1$.

Конечный детерминированный автомат (КДА) M — это такой КНА, где функция переходов δ выглядит так:

$$\delta : Q \times A \to 2^Q,$$

т.е. из любого состояния по любой букве возможен переход в точности в одно состояние - это и обеспечивает детерминированность работы такого автомата.

0.1.2 Теорема Клини

Пусть $A = \{a_0, ..., a_{n-1}\}$ - произвольный алфавит. Язык $L \subseteq A^*$ является элементом полукольца регулярных языков $R(A)$ в алфавите A тогда и только тогда, когда он допускается некоторым конечным автоматом.

0.2 Постановка задачи

Теперь сформулируем задачу в вышеуказанных терминах:

Реализовать алгоритм преобразования λ-КНА M в КНА $\hat{M}$ так, чтобы распознаваемые ими языки совпадали, т.е. $L(M) = L(\hat{M})$.

0.3 Алгоритм детерминизации автомата

Алгоритм детерминизации осуществляется в два шага - удаление λ-переходов (то есть переход от λ-КНА к КНА) и непосредственно детерминизация КНА (КНА $\rightarrow$ КДА).

0.3.1 Удаление λ-переходов

Для любого λ-КНА можно построить эквивалентный ему КНА (Здесь и далее под эквивалентностью автоматов подразумевается совпадение распознаваемых ими языков). Доказательство этого утверждения входит в обоснование алгоритма детерминизации, который описан в учебниках [1]. Основная идея этого шага такая - из состояния q_i существует переход по букве a в состояние q_j тогда и только тогда, когда выполняется одно из двух условий:

- $q_j \in \delta(q_i, a)$;
- $\exists (q_1, q_2, \ldots, q_p) \in \bigcup\limits_{p=1}^{n} Q^p$:
  $(q_1 \in \delta(q_i, \lambda)) \wedge (\forall k = \overline{1, p-1})(q_k \in \delta(q_{k+1}, \lambda)) \wedge (q_p in \delta(q_i, a))$.

То есть, если из состояния q_i какая-то цепь пустых переходов ведет в состояние, из которого есть переход в q_i по букве a, то в новом КНА будет переход из q_i в q_i по букве a.

0.3.2 Детерминизация КНА

Теперь у нас есть КНА, который всегда можно детерминизировать (это утверждение тоже есть и доказано в учебнике).

Алгоритм интуитивно следует понимать так: если моделировать КНА на компьютере, то прежде всего придется реализовать автомат так, чтобы он мог находиться одновременно в различных состояниях, в которые он может попасть по прочитанному подслову входной последовательности. Точно так же и тут - пусть в КДА состояния будут в биективном отношении с совокупностью подмножеств состояний входного КДА. Такие состояния результирующего КДА будем называть гиперсостоянием (исключительно ради удобства, в коде, реализующем алгоритм, они так и называются). Помимо этого, будем говорить, что состояние входит в гиперсостояние. если это состояние входит в то подмножество Q, в котором гиперсостояние находится в биективном отношении.

Теперь вполне разумна мысль. что переход из одного гиперсостояния в другое по букве возможен тогда, когда хотя бы из одного состояния в первом гиперсостоянии есть переход по этой букве в состояния из второго гиперсостояния. Вообще и обратное следствие верно, но доказывать его сложнее.

В результирующем автомате гиперсостояния связаны со множеством 2^Q инъективным отношением, поэтому в новом КДА необходимо определить функцию переходов следующим образом:

$$\hat{\delta} : S \times A \to S, \text{ где } \exists \xi : S \to 2^Q - \text{инъекция.}$$

Так как уже невозможно это произносить, читать и, поверьте мне, печатать, перейдем непосредственно к алгоритму.

Вход: $M = (A, Q, q_0, F, \delta)$;
Выход: $\hat{M} = (\hat{A}, S, s_0, F_s, \hat{\delta})$;

1: $\hat{A} := A$; // *Алфавиты совпадают*
2: $s_0 := \{q_0\}$ // *Входное гиперсостояние соответсвует входному состоянию исходного автомата*
3: $F_s = \emptyset$ // *Множество выходных гиперсостояний*
4: $DET = \emptyset$ // *Множество уже детерминизированных гиперсостояний*
5: $NONDET = \{s_0\}$ // *Множество еще недетерминизированных гиперсостояний*
6: **пока** $NONDET \neq \emptyset$
7:    $state := choice(NONDET)$; // *Выбираем како-либо элемент из множества и извлекаем его*
8:    **для всех** $a \in A$
9:       $next := \emptyset$; // *Новое гиперсостояние при переходе по букве a*
10:       **для всех** $q \in state$
11:          $next := next \cup \delta(q, a)$;
12:       $\hat{\delta}(state, a) := next$; // *Добавляем соответствующий переход по букве*
13:       **если** $next \notin DET$ **то**
14:          $NONDET := NONDET \cup \{next\}$; // *Если гиперсостояние не было детерминированно, то добавляем его в множество ожидающих детерминизацию*
15:       **если** $state \cap F \neq \emptyset$ **то**
16:          $F_s := F_s \cup \{state\}$; // *Если хотя бы одно состояние гиперсостояния было выходным, то все гиперсостояние - выходное*
17: $S := DET$;
18: $\hat{M} = (\hat{A}, S, s_0, F_s, \hat{\delta})$; // *Возвращаем детерминированный автомат*

0.3.3 Оценка сложности алгоритма

Очевидно, самая сложная часть этого алгоритма - детерминизация КНА. Количество гиперсостояний органиченно сверху числом 2^n, так как различных гиперсостояний ровно столько, сколько различных подмножеств n-элементного множетсва состояний Q. То есть в худшем случае предстоит детерминизировать $O(2^n)$ гиперсостояний.

При каждой детерминизации гиперсостояния алгоритм проверяет по всем буквам (8 строка алгоритма) и по всем состояниям, которые оно в себе содержит, переходы, а затем объекдиняет их с уже выявленными для предыдущих букв, то есть сложность такого шага $O(m)O(n)O(n) =$

$O(mn^2)$ (это явно сложнее чем еще одно пересечение (15 строка) и объединение (16 строка), так что это и будет сложность шага.

Итого $O(2^n)$ шагов сложностью $O(mn^2)$, итого $O(mn^2 2^n)$.

0.4 Реализация алгоритма детерминизации автомата

Ниже указаны лишь те моменты реализации автоматов и алгоритмов на языке c++ (стандарт c++17), которые действительно заслуживают внимания.

0.4.1 Кодирование автоматов (реализация классов)

Итак, автомат - это кортеж некоторых математических объектов, 2 из которых - конечные множества, 1 - подмножество одного из ранее указанных множеств, 1 - элемент множества и какая-то функция, множеством значений которой есть булеан конечного множества. Ясно, что придется кодировать множеста и подмножества. Ситуация осложняется тем, что вообще, и множеством входных символов, и множеством состояний могут быть что угодно. Даже думать не надо, чтобы построить автомат, алфавит которого - разные виды верблюдов, автомат бы распознавал регулярные каравны, а множеством состояний этого автомата могут быть и другие автоматы. Маразм, одним словом, но математика этого делать не запрещает. Однако, на алгоритме детерминизации это абсолютно никак не скажется. Нас не интересует, чем являются состояния и буквы алфавита, мы лишь требуем от этих множеств следующее:

- множество должно быть конечным;
- должна быть возможность итерироваться (пробегаться) по этому множеству;
- должна быть возможность работать с подмножествами этих множеств.

Значит, множества можно закодировать одним числом - его мощностью n. При этом, если нужно выделить отдельный элемент, то это тоже будет число i - индекс элемента такой, что $i \in Z_n$.

0.4.2 Реализация алгоритма удаления λ-переходов

0.4.3 Реализация алгоритма детерминизации КНА

0.4.4 Уязвимые для критики места

0.5 Тестирование

0.5.1 Unit-тестирование

0.5.2 Умное тестирование

0.6 Использование алгоритма

0.6.1 Формат файла-автомата (.fsa)

0.6.2 Компиляция и запуск основной программы

0.7 Заключение

~~Реализовано все круто, добавить нечего, почаще бы так писали код.~~

Bibliography

[1] А. И. Белоусов, С. Б. Ткачев. Дискретная математика. Издательство МГТУ им. Н. Э. Баумана, Москвв, 2002.