Документация и теория к коду

Супрун Артём Сергеевич

14 апреля 2025 г.

1 Введение

В данном документе представлена подробная теория и документация к коду на языке `C`, который демонстрирует работу со строками и реализации конечных автоматов для распознавания строк над алфавитом $\{a, b\}$. Автомат принимает строку, если она содержит ровно два символа `'b'` (при этом любые символы `'a'` допускаются в любом количестве и на любых позициях, за исключением нарушения условия). Документация разобрана на несколько частей: функции для работы со строками, две реализации конечного автомата (на основе конструкции `switch` и таблицы переходов) и соответствующие теоретические замечания о ДКА (детерминированном конечном автомате).

2 Работа со строками

Код содержит две вспомогательные функции для обработки строк: вычисление длины строки и удаление завершающего символа новой строки.

2.1 `my_strlen`

Функция `my_strlen` вычисляет длину строки, используя арифметику указателей. Идея заключается в следующем:

1. Определяем указатель `p`, который начинает с начала строки `s`.
2. Итеративно перемещаем указатель, пока не встретим терминальный символ `'\0'`.
3. Разность между указателем `p` (на конце строки) и начальным указателем `s` соответствует количеству элементов в строке.

Код функции:

```
size_t my_strlen(const char *s) {
    const char *p = s;
    while (*p) {
        ++p;
    }
    return (size_t)(p - s);
}
```

Листинг 1: Функция вычисления длины строки

2.2 `remove_newline`

Функция `remove_newline` удаляет завершающий символ новой строки, если он присутствует. Сначала вычисляется длина строки, затем проверяется, является ли последний символ символом новой строки (`'\n'`). Если условие выполнено, символ заменяется на терминальный символ `'\0'`:

```
void remove_newline(char *s) {
    size_t len = my_strlen(s);
    if (len > 0 && s[len - 1] == '\n') {
        s[len - 1] = '\0';
    }
}
```

Листинг 2: Функция удаления символа новой строки

3 Реализация конечного автомата

Код содержит две реализации конечного автомата для распознавания строк, составляющих язык с алфавитом $\{a, b\}$, где строка принимается, если содержит ровно два символа `'b'`.

3.1 Автомат на основе конструкции `switch`

Для реализации автомата с использованием конструкции `switch` определено перечисление `state_switch_t`:

- `STATE_INIT` — начальное состояние.
- `STATE_ONE_B` — состояние после первого вхождения символа `'b'`.

- `STATE_TWO_B` — принимающее состояние (после второго `'b'`).
- `STATE_REJECT` — состояние отклонения.

Функция `next_state_switch` осуществляет переход между состояниями по следующим правилам:

- В `STATE_INIT` символ `'a'` не изменяет состояние, а `'b'` переводит его в `STATE_ONE_B`.
- В `STATE_ONE_B` символ `'a'` оставляет состояние без изменений, а `'b'` переводит в `STATE_TWO_B`.
- В `STATE_TWO_B` символ `'a'` оставляет состояние неизменным, а любой другой символ (в частности, `'b'`) приводит к состоянию `STATE_REJECT`.

Код функции переходов:

```
state_switch_t next_state_switch(state_switch_t state,
    char c) {
     switch (state) {
         case STATE_INIT:
             if (c == 'a') {
                 return STATE_INIT;
             } else if (c == 'b') {
                 return STATE_ONE_B;
             }
             break;
         case STATE_ONE_B:
             if (c == 'a') {
                 return STATE_ONE_B;
             } else if (c == 'b') {
                 return STATE_TWO_B;
             }
             break;
         case STATE_TWO_B:
             if (c == 'a') {
                 return STATE_TWO_B;
             }
             break;
         default:
             break;
     }
     return STATE_REJECT;
}
```

Листинг 3: Функция перехода для автомата на основе `switch`

Функция `check_string_switch` проходит по вводимой строке, обновляя состояние автомата после обработки каждого символа. Если в процессе проверки встречается состояние отклонения, функция немедленно завершает работу, возвращая 0. Строка принимается только, если финальное состояние равно `STATE_TWO_B`:

```
int check_string_switch(const char *s) {
    state_switch_t state = STATE_INIT;
    while (*s) {
        state = next_state_switch(state, *s);
        if (state == STATE_REJECT) {
            return 0;
        }
        s++;
    }
    return (state == STATE_TWO_B);
}
```

Листинг 4: Проверка строки с использованием автомата на основе `switch`

3.2 Автомат на основе таблицы переходов

Вторая реализация использует таблицу переходов, что позволяет компактно задать все правила. Для этого определено перечисление `state_table_t`:

- `S0_T` — начальное состояние.
- `S1_T` — состояние после первого вхождения символа `'b'`.
- `S2_T` — принимающее состояние (после второго `'b'`).
- `REJECT_T` — состояние отклонения.

Таблица переходов представлена в виде двумерного массива:

```
static const int transition[4][2] = {
    [S0_T]      = { S0_T,      S1_T },
    [S1_T]      = { S1_T,      S2_T },
    [S2_T]      = { S2_T,      REJECT_T },
    [REJECT_T] = { REJECT_T, REJECT_T }
};
```

Листинг 5: Таблица переходов автомата

При этом индекс 0 соответствует символу `'a'`, а индекс 1 — символу `'b'`. Функция `check_string_table` проверяет строку, обновляя состояние по

заданной таблице. Если встречается недопустимый символ или автомат переходит в состояние `REJECT_T`, происходит немедленный выход:

```
int check_string_table(const char *s) {
    state_table_t state = S0_T;
    while (*s) {
        int symbol_index = -1;
        if (*s == 'a') {
            symbol_index = 0;
        } else if (*s == 'b') {
            symbol_index = 1;
        } else {
            return 0;  //
                                .
        }
        state = transition[state][symbol_index];
        if (state == REJECT_T) {
            return 0;
        }
        s++;
    }
    return (state == S2_T);
}
```

Листинг 6: Проверка строки с использованием таблицы переходов

3.3 Сравнение подходов

Оба метода реализуют один и тот же конечный автомат, принимающий строки, содержащие ровно два символа `'b'`:

- Реализация **на основе `switch`** явно задаёт переходы между состояниями. Такой подход может оказаться более понятным с точки зрения пошаговой логики, но при росте числа состояний становится менее масштабируемым.
- Реализация **на основе таблицы переходов** позволяет компактно описать все возможные переходы. Это упрощает модификацию автомата и делает код более наглядным при работе с большим количеством состояний.

4 Функция `main`

Функция `main` объединяет все компоненты программы:

1. Запрашивается ввод строки с клавиатуры.
2. Считывается строка функцией `fgets` с размером буфера 256 символов.
3. С помощью функции `remove_newline` удаляется символ новой строки, если он присутствует.
4. Строка проверяется двумя способами: сначала с использованием конструкций `switch`, затем с таблицей переходов.
5. В зависимости от результата выводится сообщение о том, принята ли строка.

Код функции:

```
int main(void) {
    char input[256];

    printf("                              : ");
    if (fgets(input, sizeof(input), stdin) == NULL) {
        fprintf(stderr, "                          !\n");
        return 1;
    }

    remove_newline(input);

    if (check_string_switch(input)) {
        printf("
           (                      switch)\n");
    } else {
        printf("
           (                      switch)\n");
    }

    if (check_string_table(input)) {
        printf("
           (                                  )\n");
    } else {
        printf("
           (                                  )\n");
    }

    return 0;
```

```
25 }
```

Листинг 7: Функция `main`

5 Теоретическое обоснование: конечный автомат

Детерминированный конечный автомат (DFA, Deterministic Finite Automaton) состоит из следующих элементов:

- **Множество состояний** Q: в нашем случае, для реализации на `switch` это {STATE_INIT, STATE_ONE_B, STATE_TWO_B, STATE_REJECT}, а для таблицы переходов — {S0_T, S1_T, S2_T, REJECT_T}.
- **Алфавит** Σ: набор допустимых символов, здесь это $\{'a', 'b'\}$.
- **Функция переходов** δ: правило, по которому автомат переходит из одного состояния в другое, зависящее от текущего состояния и входного символа. Для первого метода реализации функция переходов задана с помощью конструкции `switch`, а для второго — через таблицу.
- **Начальное состояние** q_0: состояние, с которого начинается работа автомата (`STATE_INIT` или S0_T).
- **Множество принимающих состояний** F: в данном случае, автомат принимает строку, если финальное состояние равно `STATE_TWO_B` (или S2_T), что соответствует тому, что во входной строке ровно два символа `'b'`.

Если входная строка содержит недопустимые символы или же число символов `'b'` не равно двум, автомат переходит в состояние отклонения, и строка отвергается.

6 Заключение

Представленный код демонстрирует базовые принципы работы со строками в языке `C` (например, арифметику указателей для вычисления длины строки) и два метода реализации конечного автомата для распознавания строк над заданным алфавитом. Выбор между подходом с `switch` и таблицей переходов определяется требованиями к масштабируемости

и удобству модификации кода. Подобные реализации позволяют наглядно изучить принципы теории автоматов, применимые при разработке компиляторов, процессорах текстовых данных и в других областях информатики.